Contract #F30602-91-C-0037 Final Report

Bruce A. Draper, Allen R. Hanson, Edward M. Riseman
Dept. of Computer Science
University of Massachusetts
Amherst, MA., USA. 01003
bdraper@cs.umass.edu
(413) 545-1320

Contents

1 Introduction

Over the past twenty-plus years of computer vision research, a wide variety of algorithms have been developed to solve many visual subproblems, ranging from edge extraction to vanishing point analysis to geometric model matching. Despite these advances, however, very few systems have been built that exploit the information in images to solve practical problems. The problem is a lack of understanding of how these algorithms (and representations) should be combined; the goal of this contract was to investigate the use of machine learning techniques to automatically build executable object recognition systems out of these readily available components.

This report describes a system, called the Schema Learning System (SLS), which is the final product of three years of research under contract #F30602-91-C-0037[1]. Significantly, SLS does not try to match abstract object models directly to image data. Drawing on twenty years of computer vision research, SLS compares models to data by reasoning across multiple *levels of representation*. The computer vision literature contains many representational systems for visual data, as well as many algorithms for creating and evaluating instances of these representations. SLS integrates this research by selecting the visual procedures and representations that will best satisfy a particular goal, and building an executable control strategy for invoking those procedures to achieve the goal.

SLS's recognition strategies should be immediately useful in such emerging technologies as intelligent vehicles and flexible manufacturing systems, where predictable environments invite the use of special-purpose recognition strategies. In the future, they may also be part of general-purpose computer vision systems that rely on expectations to reduce the computational cost of vision, except in those rare cases where contextual predictions fail.

1.1 The Schema Learning System

The Schema Learning System (SLS) learns special-purpose recognition strategies from training images. A teacher provides a training signal indicating the target (or goal) in each training image. By comparing the teacher's ground-truth information about a scene with the image data, SLS learns an accurate and efficient strategy for recognizing an object or object class. This strategy is then available to application programs such as autonomous vehicles or manufacturing robots any time they need to recognise an instance of the object or object class. SLS is therefore a compile-time (or "off-line" or "batch") system that learns strategies in advance of the run-time application that will use them, as shown in Figure 1.

More specifically, SLS learns strategies to satisfy *recognition goals*. One motivation for goal-directed vision is that biological vision systems are driven by the goals and actions of an agent, so that, for example, a frog has special-purpose strategies for finding prey and detecting threats [2]. More recently this idea has been revived by others who argue that computer vision should be a purposive process by which agents extract information from the world pertinent to their goals and actions [7, 1]. In SLS, a teacher provides recognition goals specifying the objects to be recognized, along with their target representations and accuracy thresholds. For example, a goal might be to recognise the position of a building to within

[1] Most of this material is from [14].

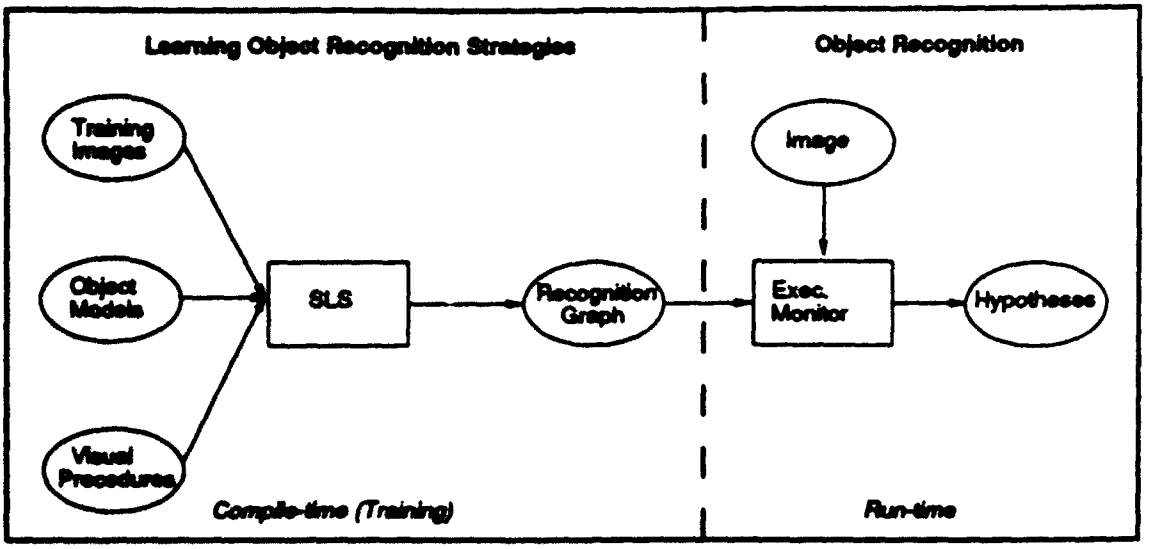

Figure 1: Top-level view of SLS architecture

5% of the distance from the camera to the object, or to identify the centroid of the image projection of a tree, plus or minus one pixel. Whatever the recognition goal, SLS's task is to learn a robust and efficient strategy for satisfying it.

1.2 The Processing Model

SLS models vision in terms of *visual procedures* (VPs) and *hypotheses*. Visual procedures are algorithms from the computer vision literature, such as region segmentation, line extraction or pose determination. VPs are thus analogous to knowledge sources in a blackboard system (e.g. [17, 13]) or Ullman's visual routines [30], in the sense that they are the procedural primitives used to build larger strategies. Hypotheses are instances of intermediate-level representations of the image and/or 3D world, including regions, line segments, coordinate transformations and object labels. At each step in the recognition process, a VP is applied to one or more hypotheses and either 1) measures features of the hypothesis or 2) generates new, higher-level hypotheses.

Recognition strategies are represented by *recognition graphs*, which are a generalization of decision trees to multiple levels of representation. Recognition graphs control hypothesis generation as well as hypothesis verification, as shown in Figure 2. The underlying premise is that image data should not be matched directly to object models. Instead, a sequence of more and more abstract descriptions of the image data, represented as intermediate-level hypotheses, are built up under constraints provided by the object model, until eventually goal-level hypotheses are generated. Recognition graphs therefore model vision as a sequence of representational transformations interleaved with hypothesis verifications. Each level of the recognition graph corresponds to one type of intermediate-level hypothesis (in blackboard terminology, one level of abstraction), and the decision tree at each level controls how

hypotheses of that type are verified. Verified hypotheses are transformed into more abstract hypotheses, until eventually goal-level hypotheses are generated. (Recognition graphs are described more thoroughly in Section 2.3.)

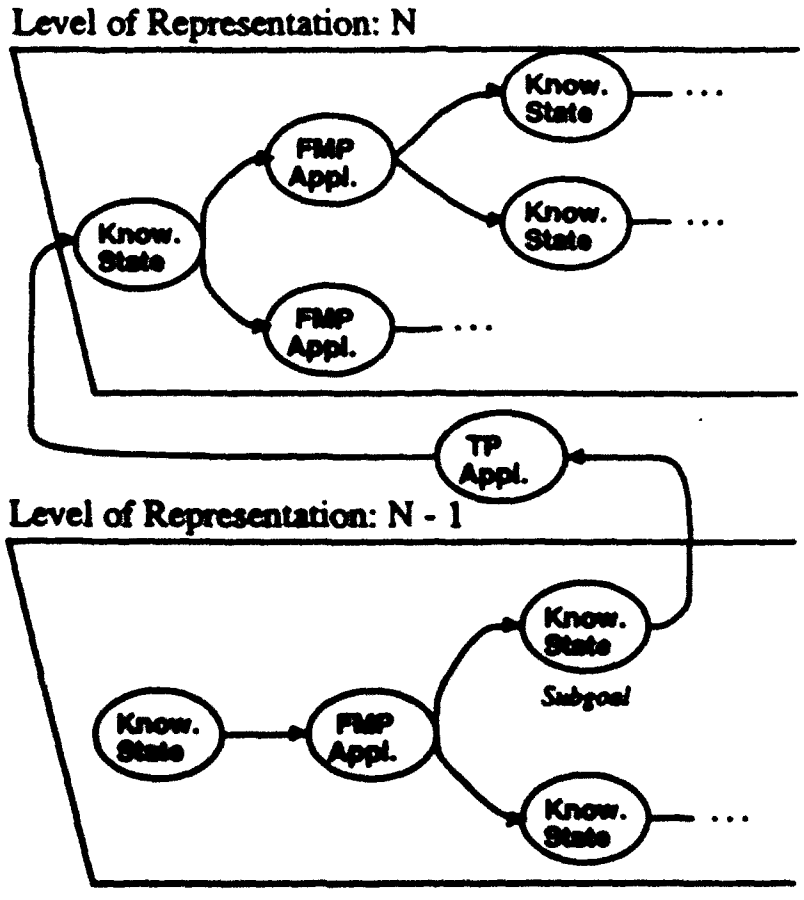

Figure 2: A recognition graph. Levels of the graph are decision trees that verify hypotheses using feature measurement procedures (FMPs). Hypotheses that reach a subgoal are transformed to the next level of representation by transformation procedures (TPs).

1.3 The Three Phases of SLS

SLS learns recognition graphs through a three step process of *exploration*, *learning from examples*, and *graph optimization*. The exploration algorithm estimates the costs and likelihoods associated with visual procedures by applying them to training images. In the process, the exploration algorithm generates more and more abstract hypotheses, eventually producing goal-level hypotheses for the learning-from-examples algorithm to analyze. The learning-by-examples algorithm inspects these examples and infers a generalized concept of how correct goal-level hypotheses are generated from images through sequences of intermediate-level hypotheses. Typically it will discover that in order to recognize an object reliably, several (possibly redundant) methods of hypothesis generation must be used.

Finally, the graph optimization algorithm creates decision trees at each level of the recognition graph that minimize the expected cost of verification. The final result is a multi-level recognition graph representing an efficient and reliable strategy for identifying an object or object class in terms of the specified goal (e.g. 2D or 3D, approximate or exact).

1.4 Contributions

The primary contribution of SLS is that it automatically learns special-purpose recognition strategies under supervision. There has been earlier work on learning shape-based recognition strategies from CAD/CAM models under known lighting conditions [20, 21, 8], and on learning to recognize two-dimensional objects from features that can be measured directly in the image, assuming known prior probabilities [9]. None of these systems, however, can do what SLS can do: learn to recognize artificial or natural objects in complex images by integrating cues from shape, color, context and other types of knowledge. SLS is able to achieve these goals because it reasons across multiple levels of representation, and takes advantage of the wealth of available computer vision procedures.

A more mundane, but still important, contribution is that SLS is the first system to supply a user (or application program) with an estimate of the expected cost of satisfying a recognition goal. This information can be critical for planning and resource allocation in robotic systems that rely on computer vision. Just as importantly, if the library of visual procedures is incapable of robustly achieving a recognition goal, SLS warns the user that the goal will not be met.

Finally, SLS gives a boost to the theory of goal-directed (purposive) vision, which has been criticized by researchers who argue that the goal of computer vision research is not just to create object recognition systems, but to put forth a coherent and parsimonious theory of vision. These researchers claim that by modeling vision as a loose (to be critical, *ad-hoc*) collection of special-purpose recognition systems, proponents of goal-directed vision abandon that goal. SLS puts forth a counterclaim by example, however; a claim that special-purpose recognition strategies do not have to be *ad-hoc* or unstructured, that they can arise through predictable and scientific mechanisms in response to a viewer's environment. Indeed, the criticism can be turned around: given that special-purpose strategies can be acquired through experience, it seems unnecessary and unjustified to assume that all visual goals must be met by a single general-purpose mechanism.

2 SLS: Representations

SLS is a compile-time (or "training-time") algorithm for learning visual control strategies under supervision. The user, acting as a teacher, provides recognition goals and training images. SLS learns to satisfy the goals by building recognition strategies that start with raw sensory data and build successively more abstract hypotheses. Hypotheses are tested at each level of representation, and verified hypotheses are used to generate new, more abstract hypotheses, eventually generating goal-level hypotheses.

2.1 The Processing Model

SLS is similar to a blackboard system in that it views recognition as a process of applying visual procedures to hypotheses. Hypotheses are representations of the image or 3D world such as points, lines, regions or surfaces; visual procedures are algorithms from the image understanding literature such as vanishing point analysis or geometric model matching. Recognition strategies take the place of dynamic schedulers in traditional blackboard systems, selecting which visual procedure(s) to apply at each step.

Therefore, recognition can be described as a branching sequence of VP invocations. The sequence begins when data arrives, typically in the form of image hypotheses[2]. Visual procedures are applied to images, producing low-level hypotheses such as points, lines or regions. New VPs are then applied to these low-level hypotheses, transforming them into more abstract hypotheses. Still more VPs are applied to these hypotheses in a repeating cycle, until eventually goal-level hypotheses are created.

2.1.1 Transformation Procedures (TPs)

Unlike most blackboard systems, however, SLS refines its processing model by dividing visual procedures into two classes, *transformation procedures* (TPs) and *feature measurement procedures* (FMPs)[3]. Transformation procedures transform old hypotheses into new hypotheses at a higher level of representation. Examples include vanishing point analysis, which creates surface orientation hypotheses from pencils of image lines, and stereo line matching, which creates world (3D) line hypotheses from pairs of image (2D) line hypotheses. Feature measurement procedures, by way of comparison, measure properties of previously existing hypotheses.

Although TPs are described as transformation operators, the word 'transformation' should not be construed as implying a one-to-one mapping between old and new hypotheses. TPs can combine information from multiple hypotheses and may generate an arbitrary number of new hypotheses. Stereo matching, for example, combines two image (2D) line hypotheses to generate a single world (3D) line hypothesis. In addition, TPs do not consume their arguments, so multiple TPs may be applied to a single hypothesis. Some readers may therefore find it helpful to think of TPs as procedures that generate new hypotheses from old hypotheses, rather than as transformation operators.

2.1.2 Feature Measurement Procedures (FMPs)

Feature measurement procedures (FMPs) calculate features of hypotheses, such as planar surface orientations and region intensities[4]. During the recognition process, many properties of a hypothesis may be uncalculated, so the set of known features describing a hypothesis

[2]Typically, but not necessarily. Active strategies may invoke procedures to acquire image data.

[3]Blackboard systems use the generic term *knowledge source* to refer to both transformation procedures and feature measurement procedures.

[4]The terms *feature*, *feature value*, *attribute*, and *property* are used inconsistently in the image understanding literature. We use the term 'property' to refer to measurable hypothesis attributes such as color, and the term 'feature' to refer to discrete measurements of those attributes, such as red.

is referred to as its *knowledge state*. Applying a FMP to one or more hypotheses computes a feature of those hypotheses, advancing them to new knowledge states. The number of knowledge states is finite, since continuous features are divided into discrete feature ranges. (Section 3.1.1 describes how continuous features are divided.)

2.1.3 VP Declarations

One of the design criteria for SLS was that it should make as few assumptions about the knowledge base as possible. SLS therefore estimates the costs and reliabilities of the VPs empirically, instead of relying on human estimates. As shown in Figure 3, the knowledge base contains only enough syntactic information to allow SLS to apply VPs to training images. In particular, for every visual procedure, the knowledge base specifies how many hypotheses are required as (run-time) arguments, the level of representation of each argument, any prerequisite features, and a lisp S-expression for invoking the VP. (Object models, if necessary, are included in the S-expression.) In addition, TP declarations include the type of hypothesis generated, while FMP declarations include the number of discrete ranges into which a continuous feature should be divided.

In addition to the generic template, Figure 3 also shows three examples of VP declarations. The first example is the vanishing point TP that creates surface orientation hypotheses from pairs of pencils by analyzing perspective distortion [11]. The next two examples show how logical dependencies are expressed in the knowledge base. The projection FMP projects boundaries of wire-frame models as image lines, given the pose of the object. The projected lines are stored in the pose hypothesis for use by other VPs, and the FMP returns a symbol declaring whether or not the object was in the field of view. The geometric matching FMP compares projected line segments to data lines, and cannot be applied to pose hypothetheses until after their projections has been computed. A prerequisite for applying the geometric matching FMP to a pose hypothesis, therefore, is that the pose has been projected and is within the camera's field of view.

2.2 Object Models

Syntactically, object models are specified as compile-time parameters to visual procedures, as mentioned above. Conceptually, however, object models should be viewed as being composed of many partial descriptions of an object residing in the system's (semi)permanent memory. Each partial description is available, both during training and at run-time, to be used as arguments to visual procedures. Models are currently implemented as compile-time parameters only because no mechanism to matching model fragments to visual procedures has been implemented.

2.3 Recognition Graphs

Interpretation strategies are represented in SLS as *recognition graphs*, which are a generalization of decision trees to multiple levels of representation. Recognition graphs control both hypothesis transformation and hypothesis verification, as shown in Figure 4. The premise

VP Declaration Template

VP Name: *VP Name*
Type: *Transformation or Verification*
Arguments: *Number of run-time arguments (hypotheses)*
Levels: *Level of Abstraction for each argument.*
Prerequisites: *List of required feature values for each hypothesis.*
Result Level: *Level of Abstraction of resulting hypotheses (GKSs only).*
Ranges: *(VKSs only) Number of discrete buckets to divide feature range into, and/or a list of ranges (if semantics are well understood), or list of possible values (if the VKS is already discrete).*
S-Expr: *Lisp expression used to invoke the KS. The symbols ih1, ih2... stand in for the run-time arguments; all compile-time parameters must be included in the lisp expression.*

Examples

VP Name: *Vanishing Point Analysis*
Type: *Transformation*
Arguments: *2*
Prerequisites: *nil*
Levels: *Pencil, Pencil*
Result Level: *Orientation*
S-Expr: *(vp:vp ih1 ih2 "camera-parameters")*

VP Name: *Line Projection*
Type: *Verification*
Arguments: *1*
Levels: *Pose*
Prerequisites: *nil*
Ranges: *Projected, Off-screen*
S-Expr: *(graphics:project ih1 "wire-frame-model" "camera-parameters")*

VP Name: *Geometric Matching*
Type: *Verification*
Arguments: *2*
Levels: *Pose, Lineset*
Prerequisites: *Projected, nil*
Ranges: *3*
S-Expr: *(geo:match-error ih1 "wire-frame-model" ih2)*

Figure 3: VP Declaration Templates. Each VP declaration in the knowledge base includes enough syntactic information for SLS to apply the VP to the training images. This includes the name of the VP, its compile-time parameters, the number of run-time arguments (hypotheses), the the level of representation (and any prerequisites) of each argument, and either the number of discrete feature ranges (for a FMP) or the type of hypothesis generated (for a TP).

behind the formalism is that object recognition is a series of small verification tasks interleaved with representational transformations. The recognition process begins by verifying low-level hypotheses. Low-level hypotheses that are verified (or at least not rejected) are then transformed into higher level hypotheses, where the verification process is repeated. The cycle of verification followed by transformation continues until goal-level hypotheses are generated and verified.

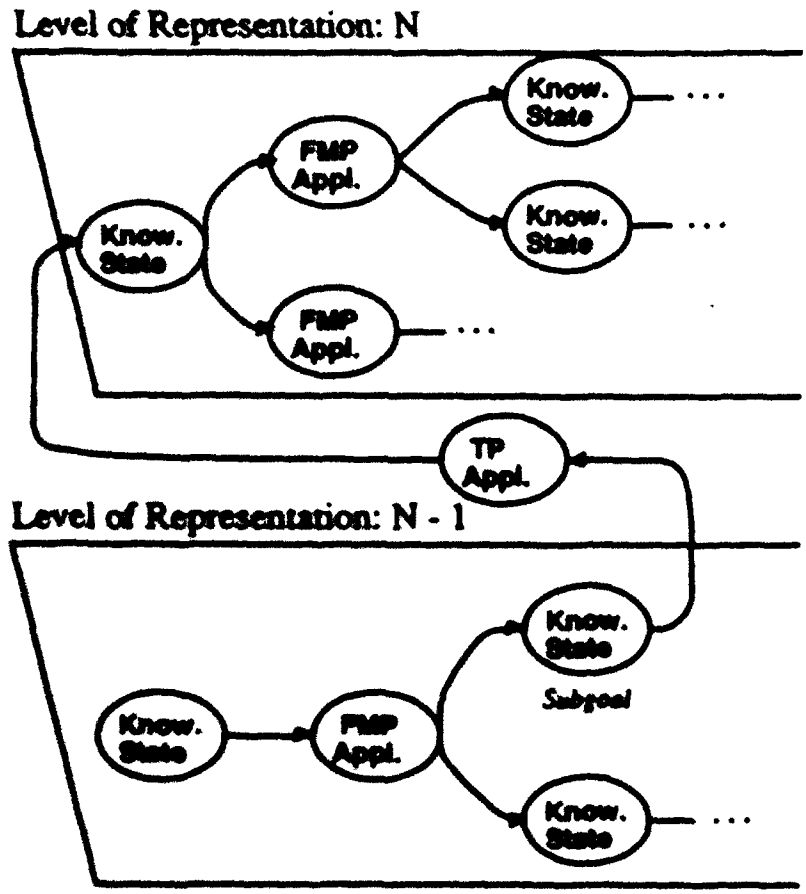

Figure 4: A recognition graph. Levels of the graph are decision trees that verify hypotheses using feature measurement procedures (FMPs). Hypotheses that reach a subgoal are transformed to the next level of representation by transformation procedures (TPs).

The structure of the recognition graph reflects the verification/transformation cycle. Levels of the graph corresponds to levels of representation, with the bottom level representing images and the top level corresponding to user's recognition goals. Levels are connected by TPs that transform hypotheses at one level into hypotheses at another. Verified goal-level hypotheses satisfy the user's recognition goal.

2.3.1 Decision Trees

Each level of the recognition graph is a decision tree directing how hypotheses at that level are verified. Borrowed from the field of operations research, decision trees are trees of alternating *choice nodes* and *chance nodes* designed to help managers make decisions about actions with uncertain outcomes ([19], Chapter 15). Choice nodes in a decision tree represent decisions over which the agent (typically a business manager) has control; chance nodes represent events the agent cannot control but whose likelihoods can be estimated. Using decision trees, managers estimate the probabilities of potential consequence of a decision or series of decisions before any action is taken. For example, a manager might consider investing in a new manufacturing facility. If the investment is made and the product sells there will be a profit, but there is some possibility that the product will not sell and the investment will be lost. This scenario can be represented by a decision tree with a choice node at the root representing the option to invest or not, and a chance node representing whether or not the product sells. In AI terminology, decision trees can be thought of as state-space representations similar to game trees with probabilistic opponents.

(Readers familiar with AI-style decision trees such as ID3 [27] will note that the choice nodes in such systems are omitted. These systems make all their choices while learning, leaving only the chances nodes in the tree. SLS does the same, leaving only one option at each choice node whenever possible. Nonetheless, it is convenient to leave the choice nodes in the formalism, both for describing the optimization algorithm that produces minimum-cost trees and for representing those situations where optimal control choices cannot be made until run-time.)

In SLS, decision trees represent the process of verifying or rejecting hypotheses. Choice nodes in the tree are hypothesis knowledge states, represented by sets of features, while chance nodes correspond to FMP invocations. The agent in this scenario is the control program that decides which feature to calculate next (i.e. which FMP to apply) based on the knowledge state of a hypothesis. The uncontrollable events are FMP invocations that return discrete features according to estimated distributions. Verification is a cycle in which the control strategy selects a FMP, the FMP returns a feature, and the control strategy selects another FMP. This cycle is represented in a decision tree as a progression from a choice node to a chance node and on to a new choice node. Eventually the process leads to a leaf node, corresponding to features that either verify or discredit a hypothesis.

Figure 5 shows a complete SLS-style decision tree. Hypotheses begin at the start state with no computed feature values, leaving the control program to choose which feature to compute. In the example shown in Figure 5 the choice is between two FMPs, A & B. Whichever FMP is selected will return a feature, advancing the hypothesis to a new knowledge state. (The reader may note that duplicate knowledge states can be joined, since the same knowledge state results from applying A and then B as B and then A. This converts SLS's decision trees into directed acyclic graphs,)

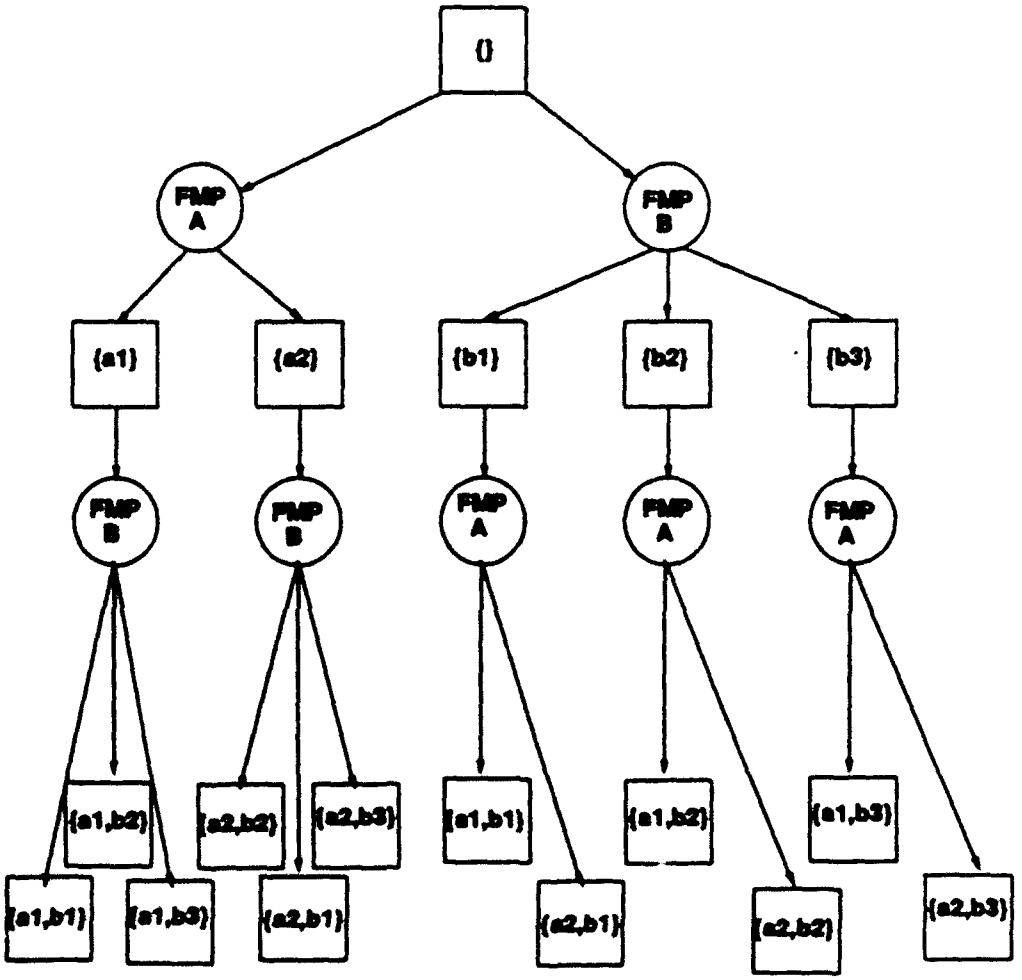

Figure 5: A Decision Tree. The squares indicate choice nodes, where the agent chooses which action to take, and the circles indicate chance nodes representing actions with probabilistic outcomes. In SLS, the agent is the run-time control program, choice nodes are hypothesis knowledge states corresponding to sets of discrete feature values, and chance nodes are FMP invocations to determine feature values. (For efficiency, the implementation joins duplicate nodes, creating a decision graph rather than a decision tree.

2.3.2 Decision Tree Optimization

Ultimately, the goal behind the decision tree formalism is not just to represent options and outcomes, but to aid in decision making. SLS constructs efficient verification strategies by determining at compile-time which options minimize the expected cost of verification. By making these decisions at compile-time, SLS eliminates the need for complex dynamic scheduling and permits the run-time control mechanism to be implemented as table-lookup.

SLS therefore decides at compile-time which FMP to apply from each hypothesis knowledge state, producing decision trees with only one option at each choice node (like the one in Figure 6). One of the options considered for each knowledge state is to stop and either accept or reject the hypothesis as it is. For hypotheses below the goal level of representation, the decision is equivalent to chosing whether or not a hypothesis should be transformed to a higher level of representation. When SLS learns to generate hypotheses it associates preconditions with each TP for selecting which hypotheses should be transformed. The preconditions are hypothesis features, and once the corresponding properties have been computed there is no reason to apply more FMPs to a hypothesis. For example, in Figure 6 we assumed that the preconditions for transforming a hypothesis were the features a1 (computed by FMP A) and b1 (computed by FMP B). Therefore, any hypothesis with feature values a2, b2 or b3 can be rejected, since they cannot lead to a goal state. SLS selects which feature to compute first by choosing the FMP that minimizes the expected cost of recognition, based on the estimated costs and outcome probabilities associated with each FMP. For example, in Figure 6, SLS decided that it was more efficient to compute feature A first and then, if a1 was returned, compute feature B, rather than computing B first and then, if b1 was returned, computing A. (Section 3.3 describes the optimization algorithm in detail.)

2.3.3 Multiple-argument FMPs

The compile-time control decisions made by SLS are conditioned on the knowledge states of hypotheses. Stated informally, SLS decides "if a hypothesis reaches knowledge state X, then take action Y". To make these decisions, SLS needs to know the possible actions from each knowledge state, their costs and the likelihoods of their outcomes. When the actions are single-argument FMPs, their applicability can be determined syntactically from the knowledge base and the costs and distributions of outcomes can be estimated from the training data. When the actions are multiple-argument FMPs, however, determining their applicability at compile-time is more difficult.

The problem is that in order to invoke a multiple-argument FMP on a hypothesis, other arguments must be available. For example, a spatial relation FMP might test whether an object hypothesis is near (above, below, adjacent to) another hypothesized object of a specified type. Such FMPs cannot be applied to a hypothesis unless a second hypothesis of the appropriate type is available at run-time. Unfortunately, when making a decision of the type "if a hypothesis reaches knowledge state X...", SLS cannot know whether a second hypothesis will be available for a multiple-argument FMP (although it does estimate the probability of another argument being available). Therefore when selecting which FMP to apply from a given knowledge state, SLS chooses the FMP that minimizes the expected cost, regardless of how many arguments it takes. If the selected FMP takes a single argument,

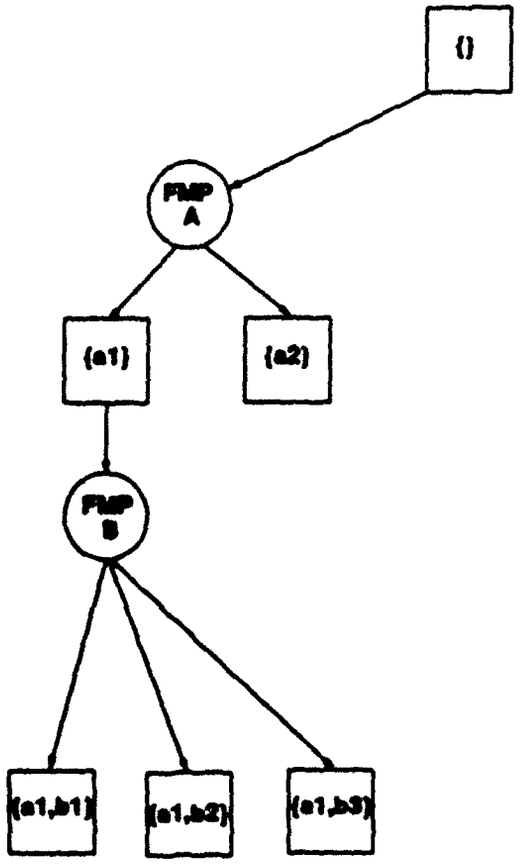

Figure 6: An SLS decision tree. SLS selects which FMP (if any) to apply from each knowledge state at compile-time, producing decision trees that have only one option at each choice node. The tree shown here is the tree SLS might build in response to the situation depicted in Figure 5, once it decided that only hypotheses with feature values a1 and b1 should be transformed to the next level of representation, and that it was more efficient to compute feature A before feature B. Note that if FMP A returns a2, then the hypothesis is rejected and no further actions are taken.

SLS knows that it can be executed at run-time and removes all other options from the choice node. If the selected FMP requires multiple arguments, however, SLS also selects a second choice, and if necessary a third choice, fourth choice, and so on, in order to ensure that at least one of the options is executable at run-time. In effect, SLS sorts the options at a knowledge state until it reaches a single-argument FMP, and the run-time control mechanism is expected to apply the highest-rated FMP whose arguments can be filled.

2.3.4 Decision Trees as Classifiers

Each level of a recognition graph can be viewed as a classifier for distinguishing hypotheses that lead to good goal-level hypotheses from those that do not. An unusual feature of these classifiers is that they are allowed to produce false positive results but not false negatives, since verifying a poor hypothesis merely causes it to be transformed to a higher level of representation and reverified, while rejecting a valid hypothesis may cause the strategy as a whole to fail. As a general rule, therefore, if the features in a knowledge base can distinguish good hypothesis from bad ones, SLS will learn highly efficient strategies. If the features are not good indicators of hypothesis reliability, on the other hand, SLS will learn a strategy that pursues many hypotheses, in order to be sure of finding a good one.

The exception to this rule is at the goal level. Depending on the application, rejecting a valid hypothesis may or may not be as damaging as verifying a false one. Consequently, the best criterion function for training a goal-level classifier is task-specific. The ideal goal-level classifier also depends on whether the recognition goal is to find a single object or to find multiple members of a class of objects. If the goal is to find a single item, no more than one hypothesis should be verified for each image, but if the goal is to find elements of a class many hypotheses may be correct.

Goal-level classification is therefore unique. When a single hypothesis is required, run-time classifiers that compare hypotheses directly to each other and select the best are used. SLS should then be viewed as a system for generating goal-level hypotheses, which are then classified by another system. When multiple goal-level hypotheses may be correct, decision trees or other classifiers that do not compare hypotheses directly to each other are more appropriate.

2.3.5 Capabilities and Limitations of Recognition Graphs

So far, object recognition has been described as a "bottom-up" process starting with an image and ending with an abstract representation of an object. Although we will continue to use bottom-up terminology, it should be noted that recognition graphs can also represent "top-down" strategies and even mixed bottom-up and top-down strategies. "Bottom-up" strategies are created from TPs that create more abstract hypotheses from less abstract ones; top-down strategies are constructed from TPs that reduce abstract hypotheses to more concrete ones. Many strategies are mixed, using TPs that produce both more and less abstract hypotheses. The only constraint enforced by SLS on recognition graphs is that the knowledge base should not contain any loops, where hypotheses of type A are created from hypotheses of type B and *vice-versa*.

At the same time, recognition graphs are not capable of representing strategies based on relative strengths of hypotheses. Traditional blackboard systems can use heuristic schedulers that apply a knowledge source to the top N hypotheses at a level of representation, but such strategies cannot be embeded in recognition graphs. Recognition graphs can represent strategies that apply VPs to hypotheses with specific sets of features, but not to the N best hypotheses in an image. (This is why a minimum distance classifier was introduced in the last section to enforce the constraint that only one goal-level hypothesis by verified per image.)

In general, "N-best" control strategies are inappropriate for multiprocessors and massively-parallel MIMD machines. To execute an "N-best" strategy, all hypotheses of a given type must be generated, and all the processors must communicate in order to compare the relative strengths of hypotheses. Only then can processing on the best hypotheses continue. "N-best" strategies are well-suited to sequential or lock-step parallel processing environments, but not multiprocessing. The recognition graph representation therefore does not support strategies that make control decisions based on the relative strengths of hypotheses.

Instead, SLS's strategies compare run-time hypotheses to training-time hypotheses. If training-time hypotheses with similar features led to correct goal-level hypotheses, then a hypothesis is pursued further; if not, it is rejected. SLS strategies base their control decisions not on the relative strengths of hypotheses from a single image, but on the relative strength of run-time hypotheses when compared to the larger pool of training hypotheses. By avoiding N-ary comparisons of run-time hypotheses (but not the low-order comparisons computed by multiple-argument VPs), SLS strategies avoid the synchronization delays and communication overhead inherent in "N best" strategies.

3 SLS: Algorithms

At the heart of SLS are algorithms that create recognition graphs from training images. As shown in Figure 7, recognition graphs are created by a three step process of *exploration, learning from examples,* and *optimization*. Speaking in general terms, the exploration algorithm generates examples of how correct, goal-level hypotheses can be generated from images through sequences of intermediate representations, and develops statistical characterisations of VPs. Generalising from these examples, the learning from examples (LFE) algorithm infers efficient methods for generating goal-level hypotheses from images. Finally, the optimisation algorithm builds a decision tree for each level of intermediate hypotheses that minimises the expected cost of verification (or rejection). Together, these algorithms produce recognition strategies that minimize the expected cost of satisfying recognition goals.

3.1 Exploration

The exploration algorithm applies visual procedures to training images and to intermediate-level hypotheses generated from training images. It begins by applying TPs to training images, producing intermediate hypotheses such as regions, lines, and points. The properties of these hypotheses are measured by FMPs, after which the hypotheses are transformed by

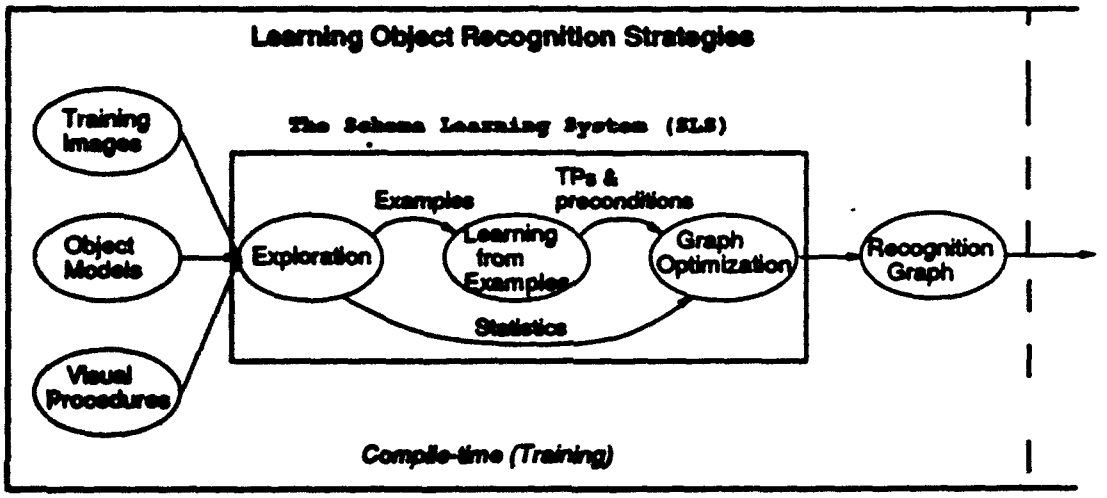

Figure 7: The Three Algorithms of SLS. This figure expands the left-hand side of Figure 1 to show the sequence of algorithms in SLS. The *exploration* algorithm creates examples of goal-level hypothesis generation and builds up statistical characterisations of TP performance. The *learning from examples* algorithm selects TPs for transforming hypotheses from one level of representation to the next and selects which features indicate that a hypothesis should be transformed, and which suggest that a hypothesis should be abandoned. The optimization algorithm builds decision trees at each level of representation that minimize the expected cost of verification.

TPs into still more abstract hypotheses. Exploration continues in this way until the supply of hypotheses that can be generated from training images is exhausted.

There are two reasons for exhaustively exploring training images. The first is to generate examples for the LFE algorithm. The training signal distinguishes correct goal-level hypotheses from incorrect ones, but it does not indicate how goal-level hypotheses should be generated from images through sequences of intermediate-level hypotheses. To learn how to generate goal-level hypotheses, SLS needs examples of how hypotheses that match the training signal can be generated. It also needs examples of intermediate-level hypotheses so that it can learn to distinguish intermediate-level hypotheses that lead to correct goal-level hypotheses from those that do not. Because the exploration algorithm exhaustively generates all possible hypotheses from training images, some of the goal-level hypotheses it generates will match the training signal, assuming there exists a strategy capable of satisfying the recognition goal. The histories of how these correct goal-level hypotheses were generated through sequences of intermediate hypotheses provide examples of how a recognition goal can be satisfied.

The second reason for exploring images is to estimate the costs and benefits of VPs in the knowledge base. In order to optimize the verification process, SLS has to know the probability of a feature given a hypothesis, as well as the expected cost of measuring that feature. Unfortunately, SLS's knowledge base does not include any information about the costs of FMPs or the probabilities of each discrete feature value. SLS therefore has to build up a statistical characterization of the FMPs by applying them to training images.

3.1.1 Discretizing Continuous Features

Once the training images have been explored, the exploration algorithm collects and processes the data. The first step is to map continuous features into discrete feature ranges. Occasionally, when the semantics of a feature are well understood, continuous features are converted into discrete values according to an explicit mapping in the knowledge base. More often, though, the relationships between features and the recognition goal are not well understood, and the discrete feature ranges are derived from the exploration data.

Ideally, a feature's range should be divided so that the resulting discrete feature values distinguish "good" hypotheses from "bad" ones. In the context of SLS, an intermediate-level hypothesis is "good" if it leads to correct goal-level hypotheses and "bad" otherwise. Good intermediate-level hypotheses are identified by finding correct goal-level hypotheses and tracing back their origins to find the intermediate-level hypotheses used to generate them. Intermediate-level hypotheses that lead to correct goal-level hypotheses are labeled as "correct", while others are labeled "incorrect".

Once hypotheses have been labeled as either correct or incorrect, SLS histograms the correct hypotheses at each level of representation, and divides the histograms of each property into overlapping ranges about the median. Each range is defined to include a fixed percentage of the samples, as shown in the top half of Figure 8. For some features, the optimal value is known to be either the minimum or maximum value, in which case the ranges are asymmetric; each range contains the optimal value plus a large enough delta to include a fixed percentage of the samples, as shown in the bottom half of Figure 8.

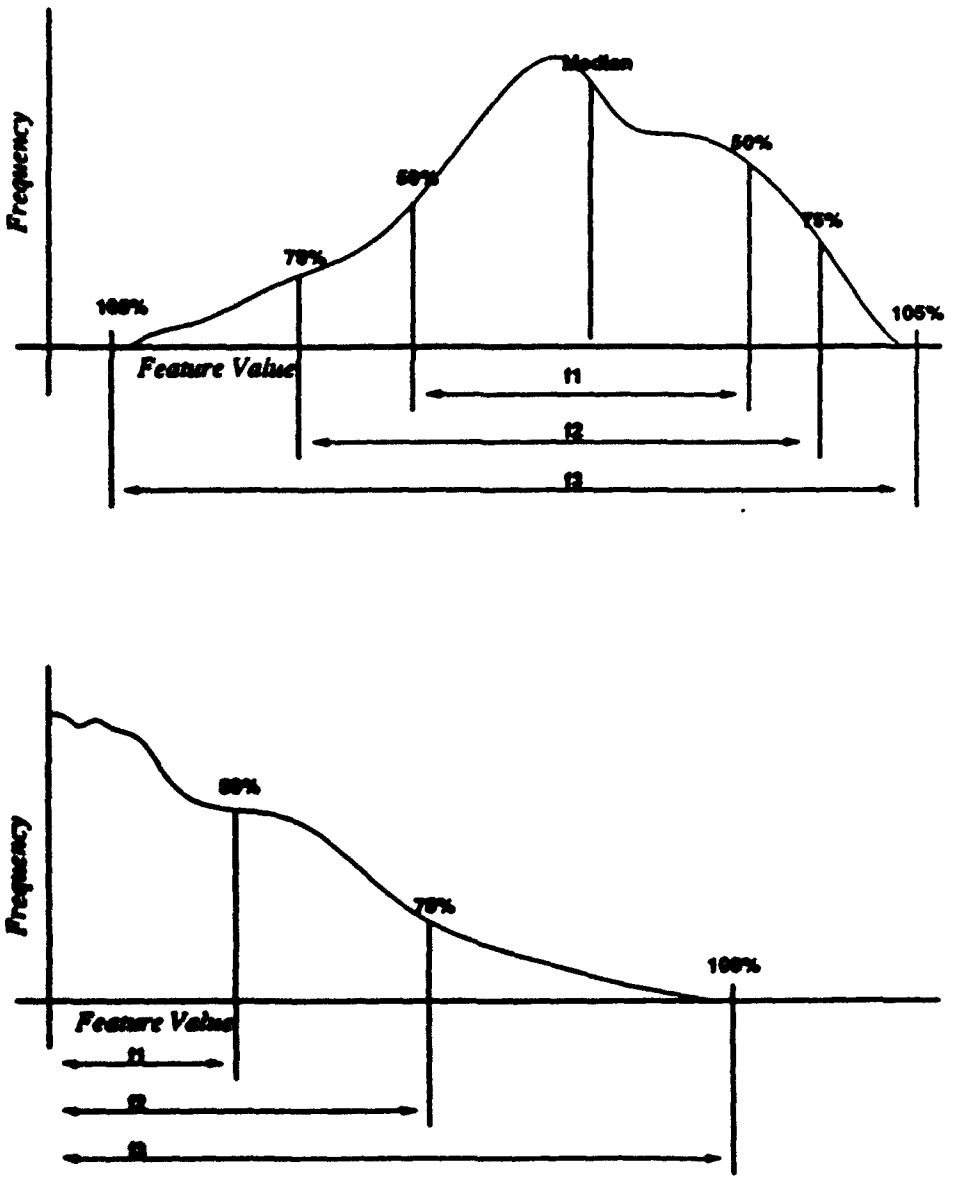

Figure 8: Discretising Continuous Features. Feature ranges are determined by histograming correct hypotheses, and selecting ranges about the median that include a fixed percentage of the samples. In the example shown at the top, 50% of all positive samples fall in the range f1, and 75% fall in f2. (f3 is a range 5% larger than needed to cover all positive samples; the extra 5% is a heuristic "fudge factor".) If the optimal value of a feature is known to be its minimum or maximum value, then the ranges are calculated from the optimum value, as shown at the bottom.

There are many other, more sophisticated, methods for dividing continuous features into discrete ranges, and no claims for the optimality of this method are made. (See Quinlan [27] for another approach.) One advantage of this method, however, is that the resulting discrete values can be interpreted probabilistically. In Figure 8, for example, the conditional probability of a correct hypotheses having feature $f1$ is .5; similarly, the probability of $f3$ is 1.0. SLS does not currently use this information, but it is helpful in trying to build an intuitive understanding of strategies learned by SLS.

3.1.2 Characterizing FMPs

Once the data has been discretized, it must be converted into a form that can be used by the LFE and optimisation algorithms. The LFE algorithm, in order to learn efficient methods for transforming images into goal-level hypotheses, needs a record of 1) the *origin* of every hypothesis generated during exploration, in terms of the TP(s) that created it and lower-level hypotheses used as arguments, and 2) the discrete features describing those hypotheses.

The optimisation algorithm, on the other hand, needs statistical models of FMP performance. Unfortunately, statistical models cannot be inferred directly from the exploration data, because the probabilities and costs associated with features depend on the quality of the hypotheses being measured. The exploration algorithm, which exhaustively explores the space of possible hypotheses, generates more hypotheses of lesser quality than SLS's run-time recognition strategy will. (After all, SLS's strategies explicitly minimise the number of false hypotheses generated.) The exploration hypotheses are in essence drawn from a different statistical distribution than the run-time hypotheses will be.

As a result, although FMP performance characterisation is conceptually part of the exploration algorithm, it is delayed until after the LFE algorithm has been run. The results of learning from examples are used to prune the exploration data, keeping those hypotheses that would be generated by the run-time strategy, and removing those that are merely artifacts of exhaustive exploration.

Once the exploration data has been pruned, the remaining hypotheses are used to characterise the performance of VPs. In particular, the exploration algorithm estimates:

- **Expected Cost (VP, F)**, the expected cost of applying a VP to a hypothesis with the feature values F;
- **Feature Likelihood (FMP, f1, F)**, the likelihood of a FMP returning feature value f1 when applied to a hypothesis with feature values F.

In general, these values are estimated from applications of FMP to similar hypotheses during training. When an insufficient number of similar hypotheses (i.e. hypotheses with feature values F) are generated during training, the dependency on F is dropped and the values are estimated across all hypotheses.

3.1.3 Making Exploration Efficient

Although SLS is designed to maximise run-time, rather than compile-time, efficiency, there may be situations where exhaustively exploring the training data is not feasible. In such

cases, the cost of exploration can be heuristically reduced by not exploring hypotheses that do not satisfy constraints derived from the goal-level solution, For example, if the recognition goal is to recover the three dimensional position of an object, any region hypotheses that do not overlap the object's projection can be rejected without being explored further. Similarly, points, lines, planes, and other types of geometric hypotheses can be rejected if they fail to overlap the correct solution or its projection. In this way, the combinatoric nature of exploration is damped, but the positive examples needed by the LFE algorithm are still generated.

The disadvantage of this heuristic is that negative examples are used in SLS 1) by the LFE algorithm, to select the minimal cost DNF subterm (see Section 3.2.3), and 2) to estimate the costs and probabilities associated with features. At the risk of a less efficient strategy, both tasks can be accomplished by exploring only a subset of negative hypotheses and extrapolating the results. However, we have not used this heuristic, preferring instead to explore the training data exhaustively, because its precise effects are hard to analyze.

3.2 Learning from Examples (LFE)

SLS's learning from examples (LFE) algorithm analyses correct hypotheses produced during exploration and infers from them an efficient scheme for generating accurate goal-level hypotheses. The approach reflects the idea that recognition is a series of transformations interleaved with verifications. By looking at the histories of how correct hypotheses develop, SLS learns how to generate goal-level hypotheses from images through series of intermediate-level hypotheses. At the same time, it learns which features of intermediate hypotheses indicate that a hypothesis should be pursued, and which imply that a hypothesis should be abandoned.

3.2.1 Learning from Examples

In the machine learning literature, the term *learning from examples* refers to algorithms that learn rules for evaluating examples. Following the terminology in the *AI Handbook* [10], learning from examples problems are defined in terms of *instance spaces* and *rule spaces*. The instance space is the set of possible examples or *instances* that might be encountered, either during training or testing. The rule space is the set of possible inference rules for evaluating instances. In general terms, learning from examples algorithms search rule spaces for the best methods of evaluating instances.

In SLS's LFE algorithm, the task is to generate correct goal-level hypotheses from images through sequences of intermediate representations. Instances are strings of hypotheses and TPs that transform images into correct goal-level hypotheses. The rule space is composed of sets of features and TPs: the features determine which hypotheses should be pursued at each level of representation, and the TPs indicate how they should be transformed. The goal of the LFE algorithm is to select sets of features (TP preconditions) and TPs that generate a correct hypothesis for every object instance in the training set, while generating as few false hypotheses as possible.

3.2.2 Dependency Trees

Inside the LFE algorithm, instances of correct hypotheses are represented as *dependency trees*. A dependency tree is an AND/OR tree recording the TPs and intermediate-level hypotheses on which a goal-level hypothesis depends. For example, a correct 3D pose hypothesis might have been generated by fitting a plane to a set of 3D line segments. If so, the pose hypothesis is dependent on the plane fitting TP and the 3D line segments, as well as the TPs and hypotheses needed to generate the 3D line segments, as shown in Figure 9. In general, dependency is recursive, with 'AND' nodes in the tree resulting from TPs that require multiple arguments (and are therefore dependent on more than one hypothesis), and 'OR' nodes in the tree occurring when more than one TP redundantly generates the same hypothesis.

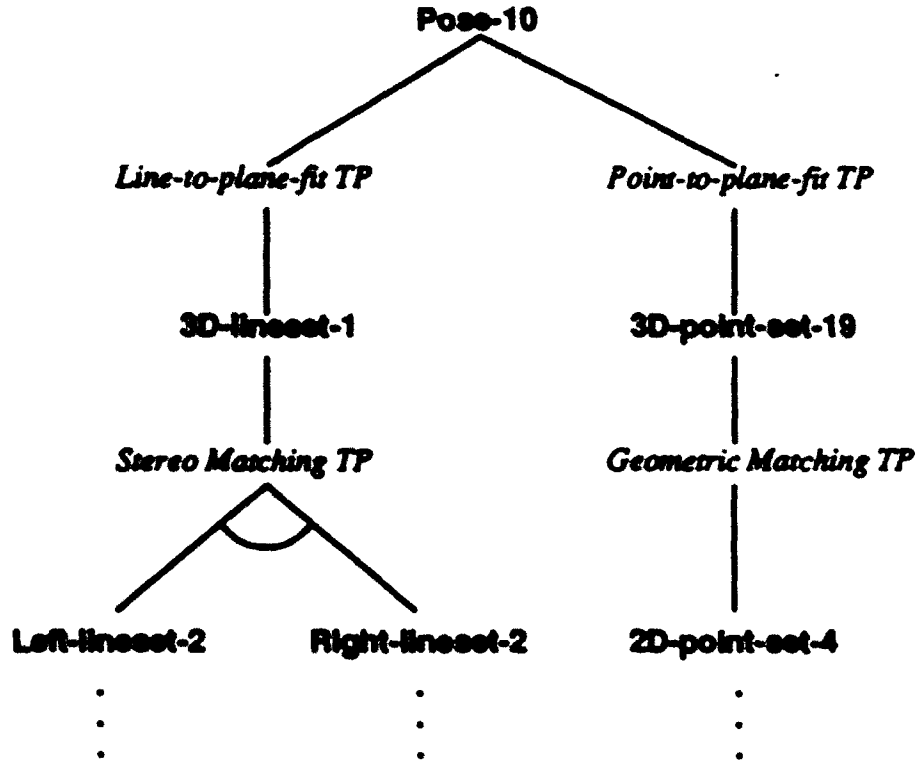

Figure 9: An example of a dependency tree showing the different ways that one correct pose hypothesis can be created during training.

Each dependency tree represents the different methods for generating a specific hypothesis. In the example in Figure 9, pose-10 can be generated either by applying the line-to-plane-fit TP or the point-to-plane-fit TP, but at least one of the two is required. Furthermore, if the line-to-plane-fit TP is used, it must be applied to 3D-lineset-1. Alternatively, if the point-to-plane-fit TP is used instead, it must be applied to 3D-point-set-19.

Dependency trees like the one in Figure 9 apply to specific hypotheses generated during exploration. The first step in inferring a more generalized scheme for transforming images

into goal-level hypotheses is to generalise the dependency trees by replacing hypotheses with their feature vectors, as shown in Figure 10. The rationale for the substitution is that TPs have preconditions associated with them that select the hypotheses to which they will be applied. If a TP needs to be applied to hypothesis H to ensure that a goal is met, then only features of H should be considered as preconditions for the TP.

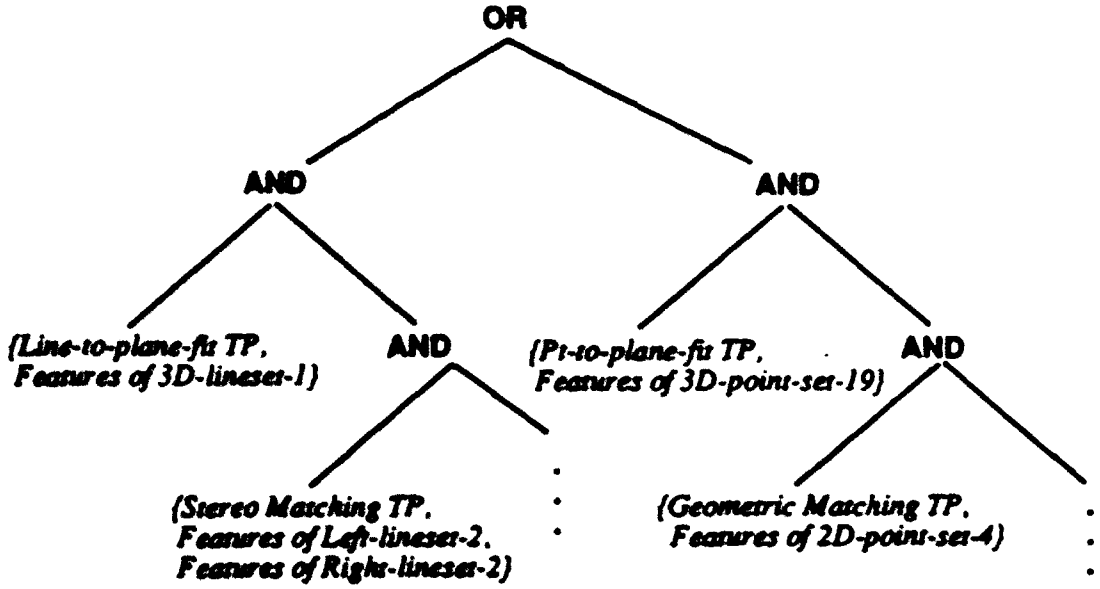

Figure 10: A generalised dependency tree created by replacing the hypotheses in Figure 4.3 with their feature values.

In general, a hypothesis is guaranteed to be created by any set of preconditioned TPs that "satisfies" its dependency tree. A dependency tree DT is satisfied by a set of TPs G (with affiliated preconditions P) if: 1) the root of DT is an AND node, and every subtree of DT is satisfied; 2) the root of DT is an OR node, and at least one subtree of DT is satisfied; or 3) the root of DT is a leaf node with TP g and preconditions P such that g is in G and the preconditions of g either match or are a superset of P.

3.2.3 LFE: A DNF-based Algorithm

The algorithm for finding optimal sets of TPs and preconditions is deceptively simple:

1. Convert the generalised dependency tree of a correct goal-level hypothesis to disjunctive normal form (DNF)[6].

2. For every other correct goal-level hypothesis:

   (a) Convert its generalised dependency tree to DNF.

[6]The disjunctive normal form of a logical expression is an OR of ANDs of monomial expressions, for example $(A \wedge B) \vee (A \wedge C)$.

(b) "AND" together the new DNF expression with the previous DNF expression.

(c) Convert the resulting 'AND' tree to DNF[6].

3. Select the conjunctive subterm that generates the fewest total hypotheses.

By the logic of the dependency relation, the TPs and preconditions in any conjunctive subterm of the final DNF expression are sufficient to re-generate the correct goal-level hypotheses from the training images. By selecting the minimal term, SLS chooses the best method for generating correct hypotheses.

AND/OR dependency trees are converted to DNF by a standard algorithm that first converts every subtree to DNF and then either merges the subterms, if the root is an OR node, or takes the symbolic cross product[7] of the subterms, if the root is an AND node. If a TP is ANDed with itself when taking the cross product, its preconditions are the intersection of the preconditions of the two instances being ANDed.

This basic algorithm is altered slightly to improve efficiency. Because SLS seeks to find the minimal term (measured as the number of hypotheses generated) of the DNF expression rather than every term, any conjunctive subterm that is a logical superset of another can be pruned, reducing the total number of terms considered. A second modification is to sort the correct goal-level hypotheses according to the size of their dependency trees and to iterate in step two from the simplest dependency trees to the most complicated. This reduces the size of the interim DNF expressions without affecting the final DNF expression.

3.3 Graph Optimization

As was stated earlier, recognition graphs interleave verification and transformation, using FMPs to measure properties of hypotheses and TPs to transform them to higher levels of representation. By building dependency trees from the training samples, converting them to DNF and picking the minimal subterm, SLS learned which TPs to use to transform hypotheses from one level to the next. Just as important, it learned which preconditions a hypothesis must meet before it should be transformed. These preconditions are the subgoals of the recognition process at intermediate levels of representation.

The optimization algorithm optimizes hypothesis verification by building decision trees for each level of representation that minimize the expected cost of reaching a subgoal or, conversely, of deciding that a hypothesis cannot satisfy a subgoal and should be rejected. The decision trees are constructed by first building a graph representing all possible sequences of FMP applications, and then optimizing the graph by determining the options at each choice node that minimize the overall cost of recognition, and removing all other options (although when multiple-argument FMPs are used, several options may be left at a choice node, sorted in terms of desirability; see Section 2.3.3). The final result is a decision tree at each level of representation that minimizes the expected cost of verification.

[6]Logically, this algorithm is equivalent to the simpler two-step process of ANDing all the dependency trees together and converting the result to DNF. However, iteratively adding each new dependency tree to an evolving expression simplifies analysis.

[7]Symbolic cross product: $\{A, B\} \times \{C, D\} = \{AC, AD, BC, BD\}$.

3.3.1 Graph Layout

For each level of representation, a directed acyclic graph is constructed representing all possible sequences of FMP applications. The graph starts from a single knowledge state, corresponding to a newly generated hypothesis for which no features have been computed. The start state, like all knowledge states, is a choice node in decision tree terminology, since a control program gets to choose which FMP to apply to hypotheses in this state. FMP applications nodes are added for every FMP that can be applied to a hypothesis in the start state. These FMP applications lead to new knowledge states, which in turn have more FMP applications attached to them, and so on. The expansion of the graph continues until it reaches either a subgoal knowledge state or a knowledge state that is incompatible with every remaining subgoal (i.e. a failure state).

For example, Figure 11 shows the initial graph for a level of representation with two FMPs and a subgoal of {a1,b1}. Graph construction begins with the start state and expands by adding a chance state for each FMP. The FMPs lead to a total of five new knowledge states, but three of them are failure states that are incompatible with the subgoal {a1,b1}. The other two states each have one more FMP to be applied, leading to four more knowledge states, one of which is the subgoal state and three of which are failure states.

More formally, we refer to subgoal states and failure states as the *terminal* states for each level of the recognition graph. The cost of promoting a hypothesis from knowledge state n to a terminal state is called the Expected Decision Cost (EDC) of knowledge state n, and the expected cost of reaching a terminal state from state n using FMP v[8] is the Expected Path Cost (EPC) of n and v. Since features are discrete, we denote the possible outcomes of a FMP v as a set $R(v)$, and the probability of a particular feature value f being returned as $P(f|v,n), f \in R(v)$.

The EDC's of knowledge states can be calculated starting from the terminal states and working backward through the recognition graph. Clearly, the EDC of a subgoal or failure state is zero:

$$EDC(n) = 0, \quad n \in \{terminal\ states\}.$$

If we limit ourselves to single-argument FMPs, the expected path cost of reaching a terminal state from a FMP application node is:

$$EPC(n,v) = C(v) + \sum_{f \in R(v)} (P(f|n,v) \times EDC(n \cup f))$$

where n is a knowledge state expressed as a set of feature values, $n \cup f$ is the knowledge state that results from FMP v returning feature value f, and $C(v)$ is the estimated cost of applying v.

The EDC of a knowledge state, then, is the smallest EPC of the FMPs that can be executed from that state:

$$EDC(n) = \min_{v \in VP(n)} (EPC(n,v))$$

[8] v is an awkward abbreviation for a feature measurement procedure, but f will be used for feature values and p would look like a probability value. Since FMPs are a subclass of VPs, v is therefore used.

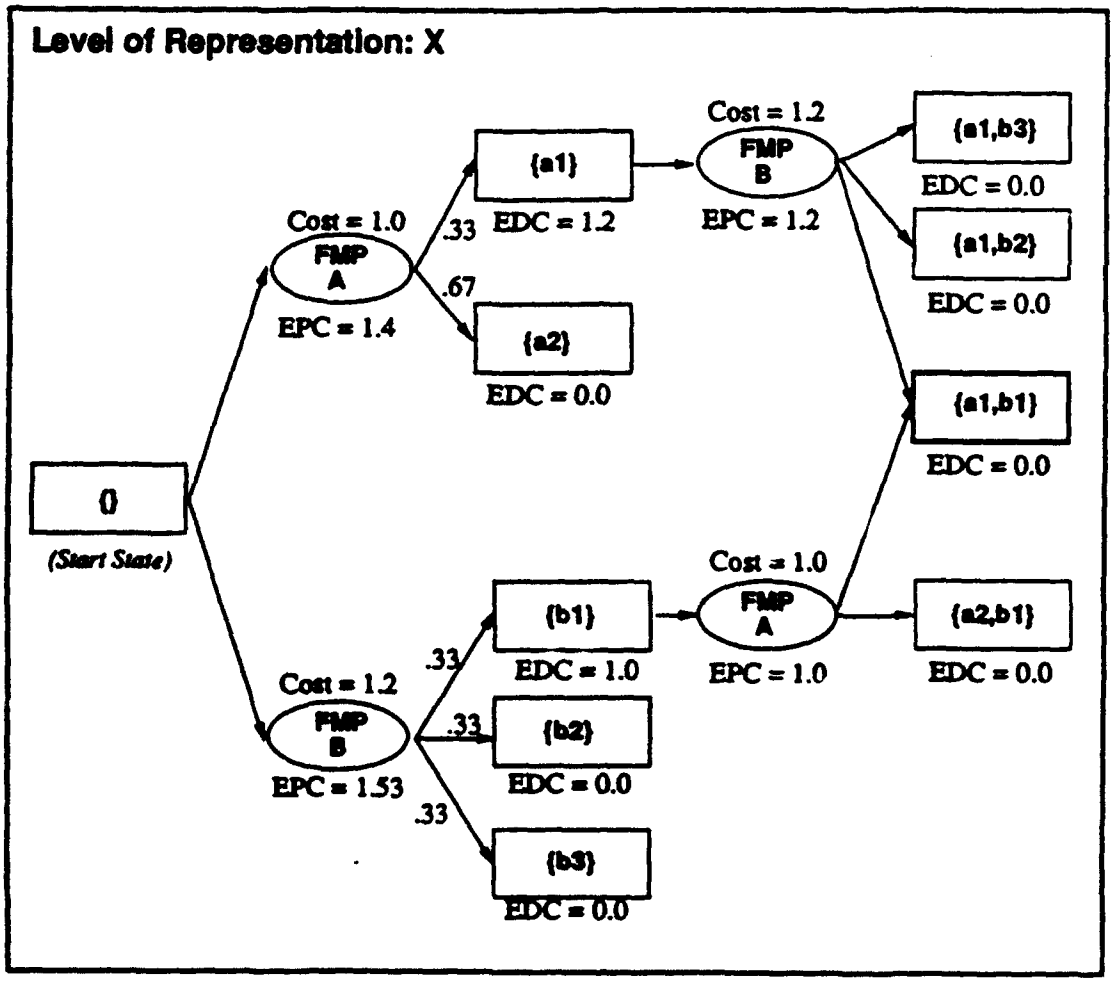

Figure 11: An initial decision graph. Choice nodes, shown as rectangles, correspond to knowledge states of a hypothesis. Chance nodes, shown as ovals, represent FMP applications. Starting from an empty knowledge state, the system adds a chance node corresponding to each FMP. Since FMPs measure feature values, they lead to new knowledge states, where new FMPs can be selected. The graph expands until it reaches either a verification state, or a state that is incompatible with the features of a verification state.

where $VP(n)$ is the set of FMPs applicable at node n. The minimal-cost decision tree is created by making a single pass through the directed acyclic graph, starting at the terminal nodes are working backward toward the start state. At each knowledge state, the pruning process calculates the EPC of every FMP that can be applied from that state, and removes all FMP application nodes except the one with the smallest EPC. The final result is the minimal-cost decision tree.

Figure 12 shows the result of pruning the initial graph shown in Figure 11. Starting at the terminal nodes and working backward, the first choice states the pruning algorithm considers are $a1$ and $b1$. These states have only one option each, however, so selecting the minimum-cost option has no effect. The next choice node encountered is the start state , where there are two options, since the system can choose to compute feature A or feature B. However, as depicted in Figure 11, the expected cost (EPC) of verifying hypotheses if feature B is computed first is 1.53, while the cost of verifying hypotheses by computing feature A first is only 1.4. Consequently, the optimization algorithm prunes option B from the start node in Figure 11, leaving the optimized decision tree shown in Figure 12.

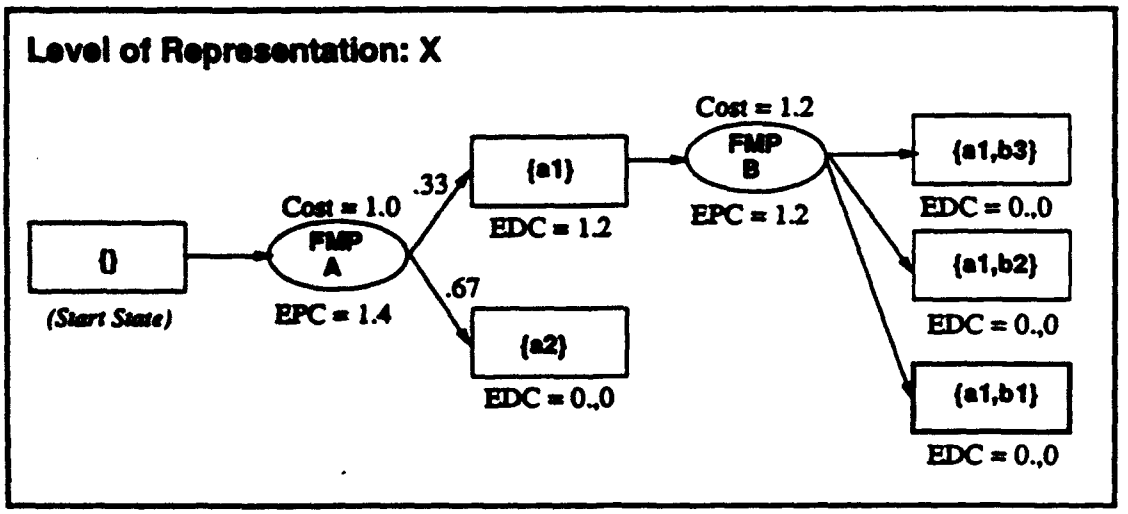

Figure 12: A pruned decision graph. This Figure shows the graph depicted in Figure 11 after it has been pruned by the graph optimization algorithm. All actions which either do not lead to the subgoal state or which are not on the most efficient path to the subgoal have been removed.

The equations above made the simplifying assumption that all FMPs were applied to individual hypotheses. The analysis gets more involved when we permit multiple-argument FMPs, such as FMPs that measure spatial or other relations between hypotheses. The problem is that the equations above implicitly assume that if FMP v is an option at knowledge state n, then v can be applied to any hypothesis reaching state n. It is always possible, for example, to measure the color of a region hypothesis or the length of a line hypothesis. With multiple-argument FMPs, however, this assumption is no longer valid, since whether or not a multiple-argument FMP can be applied to a hypothesis at a knowledge state n depends on whether the other arguments of the FMP can be filled (see Section 2.3.3).

As a result, we introduce a new term $P_{app}(v|n), v \in VP(n)$, the probability that v can be applied to a hypothesis in state n. (For single-argument FMPs, $P_{app}(v|n) = 1, \forall v \in VP(n)$.) In addition, because multiple-argument FMPs v may be applied more than once to a hypotheses by varying the other arguments, we must consider the possibility that a FMP may return a feature that had already been computed (or equivalently, may return nothing), with the result that a FMP application may not change a hypothesis' knowledge state. Under these conditions, we do not talk about the expected path cost of applying a FMP from a knowledge state (i.e. $EPC(n, v)$), but rather the expected path cost of applying a FMP from a knowledge state with a set of alternate FMPs V in reserve, in case v cannot be applied or fails to calculate a new feature.

Despite the changes, the EDC of a subgoal or failure state is still zero:

$$EDC(n, V) = 0, \quad \forall V; n \in \{terminal\ states\}.$$

In addition, because a VP application may not advance a hypothesis to a new knowledge state, we must consider the possibility of "running out" of FMPs:

$$EDC(n, \emptyset) = 0, \quad \forall n.$$

However, the expected path cost of reaching a terminal state from a FMP application node with V other FMPs in reserve is now:

$$\begin{aligned} EPC(n, v, V) = \; & P_{app}(v|n) \left[C(k, v) + \sum_{f \in R(v), f \notin n} (P(f|n, v) \times EDC(n \cup f, VP(n \cup f))) \right. \\ & \left. + \sum_{f \in R(v), f \in n} (P(f|n, v) \times EDC(n, V)) \right] \\ + \; & (1 - P_{app}(v|n)) \times EDC(n, V - v) \end{aligned}$$

The EDC of a knowledge state is still the smallest EPC of the FMPs that can be executed from that state. Minimizing the EDC of a knowledge state is no longer sufficient, however, for generating the optimal strategy. The benefit of a FMP application is the sum of the benefit it provides to each of its arguments, and the most efficient decision tree is created by selecting the FMP at each knowledge state with the highest ratio of total benefit to cost. We refer to this ratio as the gain of a FMP:

$$Gain(v, n_1, ..., n_m) = -C(v, n_1, ..., n_m) + \sum_{n} \sum_{f \in F_n} (P(f|v, n_1) EDC(n \cup F_n) - EDC(n))$$

where F_n is the set of feature values that might be returned by FMP v for argument n.

3.3.2 Estimating Total Cost

The equations above establish a mutually recursive definition of the expected decision cost of a knowledge state. The EDC of a knowledge state is the EPC of the optimal FMP application

from the state; the EPC of a FMP application is the expected cost of applying the FMP plus the expected EDC remaining after the FMP has been applied. The recursion bottoms out at terminal nodes, whose EDC is zero. Since every path through the object recognition graph ends at either a subgoal or a failure node, the recursion is well defined.

Furthermore, the total cost of recognition can be estimated from the EDCs of start states and the expected costs of the TPs selected by the LFE algorithm. The EDC of the start state for a level of representation estimates the expected cost of verifying or rejecting hypotheses at that level. By estimating the total number of hypotheses generated at each level by the preconditioned TPs and multipying it by the EDCs of the start states, the total cost of verification can be estimated. Since the expected number of times a TP will be executed can also be estimated from the LFE algorithm's results, the total expected cost of recognition can be obtained easily.

4 Experiments

We present three examples of SLS learning recognition strategies. In the first exercise, SLS learns a strategy for finding the (2D) position of a tree in images taken from an approximately known location. The second demonstration goes one step further, as SLS learns a strategy for determining the 3D pose of a building, again from an approximately known viewpoint. Finally, in the third exercise, SLS learns to recognize another, more complex building from an arbitrary position on the ground plane.

In addition to demonstrating that SLS can learn recognition strategies for complex vision applications, these exercises are designed to show that SLS can recognize both natural and man-made objects, can recognize them from either known or unknown viewpoints, and can do so in either two dimensions or three. SLS is therefore general enough to support a wide range of vision applications.

4.1 Implementation Notes

For these demonstrations, SLS was implemented in Common Lisp for a TI Explorer II Lisp Machine, as was the library of visual procedures. Hypotheses were tokens in ISR, a database system designed for computer vision applications [6]. All pictures were taken with a 35mm camera and digitized on an Optronix Colormation C4500 Digitizer/Photowriter.

The demonstrations are unfortunately limited by the inefficiency of the implementation. No effort was made to optimize either the visual procedures, which account for most of the source code, or SLS itself. As a result, exploring a single image can take on the order of two hours. The rest of SLS's processing, including all of the learning from examples and optimization procedures, takes approximately another hour. It was therefore impossible, as a practical matter, to run experiments with more than about twenty training images, particularly considering that lisp machines have no batch processing facilities.

Training set size in turn limits the robustness of the strategies SLS can learn. This is particularly a problem in the third demonstration, where the task is to learn the 3D pose of a complex object from an arbitrary viewpoint. Since the training set includes only a few

examples from each generic view, the resulting recognition strategies are less robust than those learned from twenty examples of a single view.

4.2 Tree Recognition from an Approximately Known Viewpoint

In the first demonstration, SLS learns a strategy for recognizing a tree from an approximately known viewpoint. The strategy is not required to recognize all trees, but rather a specific tree that serves as a landmark, in this case the tree behind the telephone pole in Figure 13. The goal of the strategy is to determine the image position of the tree for triangulation, and in particular the horizontal coordinate of the center of the tree.

4.2.1 Training Images

The training data is selected from a set of twenty-one images collected along a hundred foot stretch of a footpath on the UMass campus. Figures 13 and 14 show the first and last images of the sequence. The images were taken level to gravity ($\pm 1°$) and from approximately four feet above the ground, although the ground rises and falls over the course of the sequence. The camera was also subjected to small rotations in pan from one image to the next. As a result, the pose of the camera has four degrees of freedom, with large variations in position in the ground plane and smaller deviations in camera height and pan.

4.2.2 Recognition Goal

Since the conceptual goal is to find the center of the tree, the user must specify a recognition goal that conveys this information. One possibility is to represent tree projections as image regions, with the centroid of each region representing the center of the tree. If the tree is partially obscured, however, the centroid of the region will not correspond to the center of the tree. A better representation for determining the center of a tree is to represent the boundary of a tree's projection in the image as a parabola, with the locus of the parabola corresponding to the center of the tree, as in Figure 15. The selected recognition goal is therefore to generate and verify parabola hypotheses whose locus is within three pixels of the projected center of the tree. The training signal was the position of the center of the tree in each image, as determined interactively by the user with a mouse.

4.2.3 Testing Methodology

Because of the relatively small size of the training set, SLS was tested with a "leave one out" methodology, in which strategies are trained on twenty images and tested on the twenty-first. The process is repeated twenty-one times, each time with a different image "left out" of the training set and used as the test image. Each trial tests whether a strategy learned over twenty training images satisfies the recognition goal on the twenty-first. In addition to testing for robustness, the suite of twenty-one trials also tests SLS's ability to predict the reliability and average cost of its strategies.

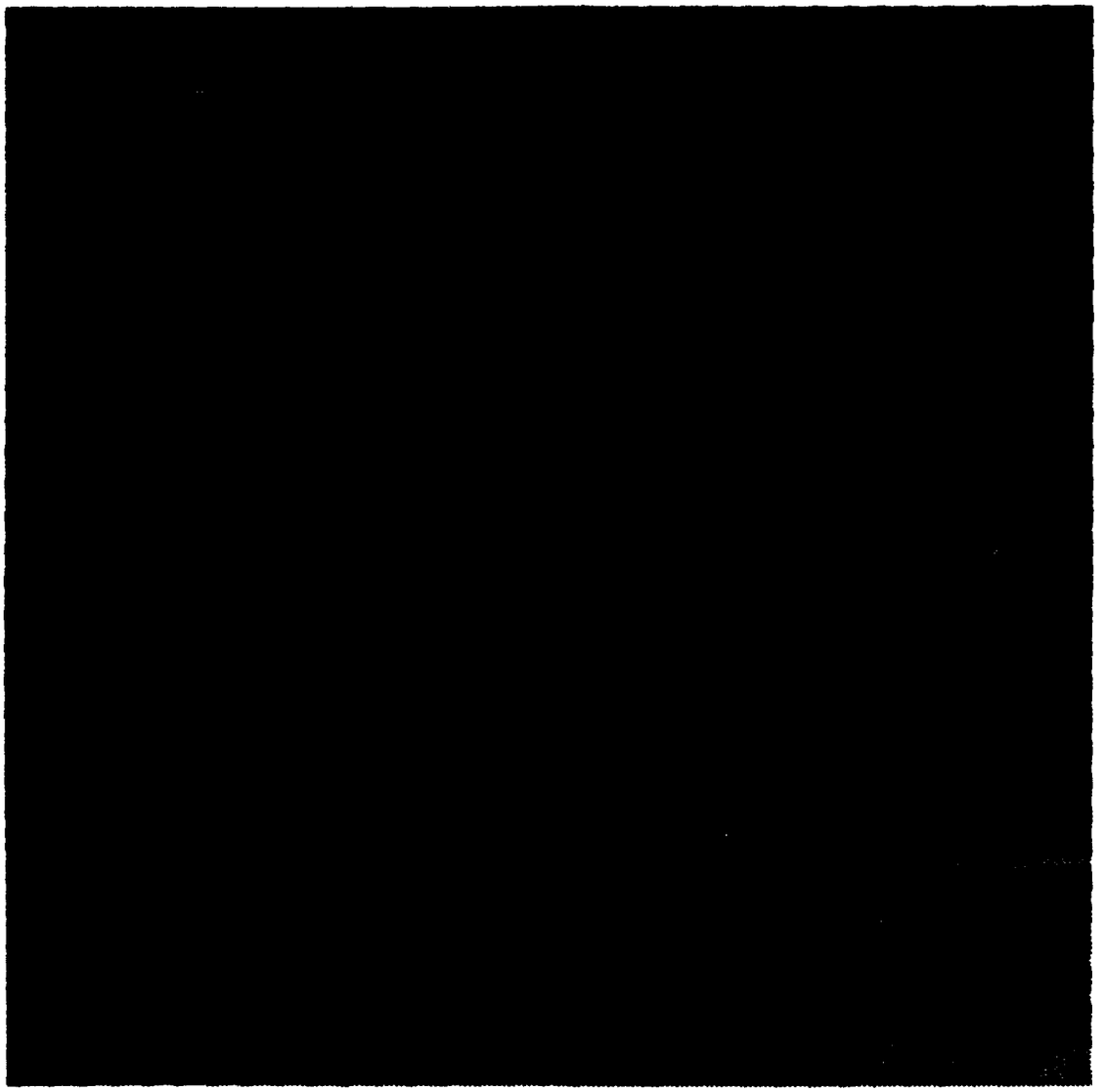

Figure 13: The first of twenty-one training images. The images were taken along a hundred-foot section of the path, with the camera level to gravity.

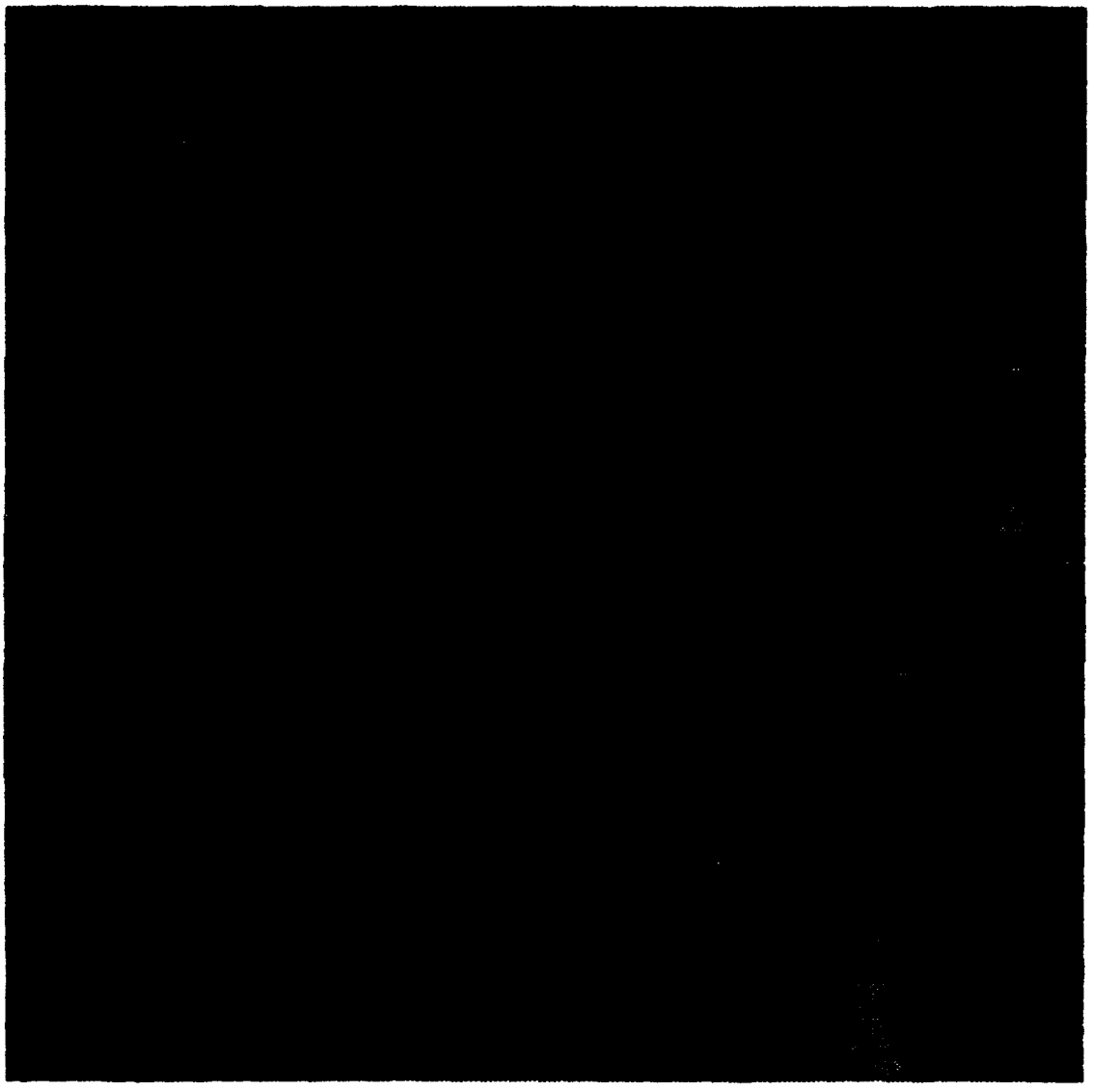

Figure 14: The last of twenty-one training images. The pose of the camera has four degrees of freedom, with large variations in position in the XZ (ground) plane and small differences in camera height and pan.

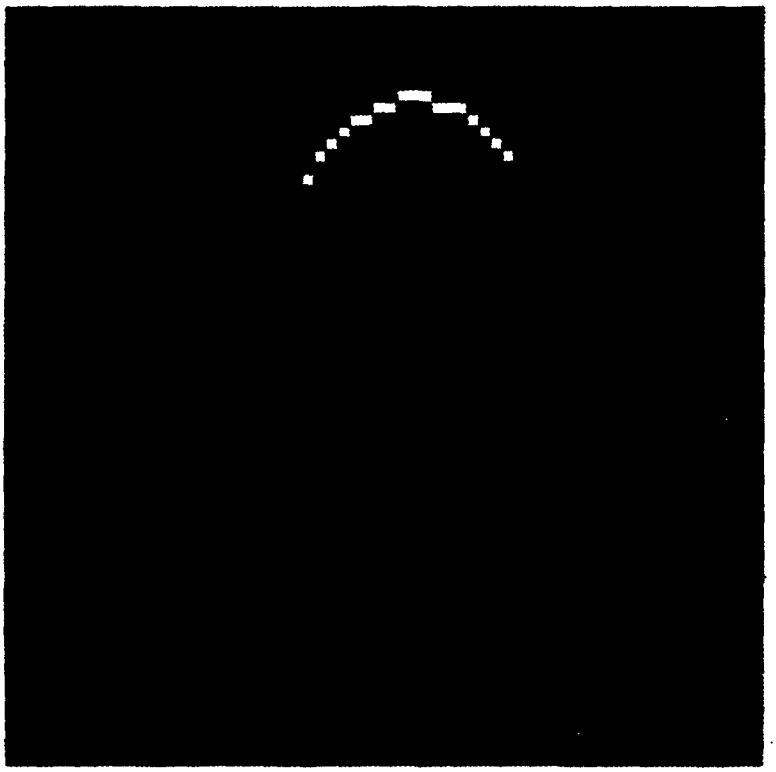

Figure 15: Representing the 2D projection of a trees as a parabola in the image. This figure shows a piece of the image in Figure 14, including the landmark tree, with a parabola hypothesis representing the location of the tree.

4.2.4 The Knowledge Base

The two-dimensional tree recognition task has the simplest knowledge base of the three exercises. Apart from the images themselves, only a handful of visual procedures and three types of representations – regions, sets of regions, and parabolas – are used. Figure 16 is an idealized depiction of the tree recognition knowledge base. The lowest two levels contain sets of regions, with the bottom level holding image segmentations and the second level storing sets of green, highly textured regions. The third level holds region hypotheses that have been pieced together from the sets of fragmented regions on level two, while the fourth level is for smoothed regions. Finally the top (goal) level is for parabola hypotheses, which may have been fit to either the rough regions on level three or the smoothed regions on level four.

It is notable that sets of hypotheses can be hypotheses themselves. One motivation for reasoning about sets is that sets may have properties not possessed by any of their members. Although this is not a factor in tree recognition, upcoming exercises will reason about *pencils* of lines, which are sets of lines that meet at a common point of intersection. The so-called "vanishing point" is a property of the set of lines that is not a property of any of the individual line segments. The other reason for using sets, and the one of concern here, is efficiency. Although it is possible to segment the image and reason about every region independently, doing so would generate several hundred hypotheses. It is far more efficient to reason about the segmentation as a single hypothesis, and create TPs that select relevant regions from it.

4.2.5 Reliability Results

The most basic question concerning SLS is whether the strategies it learns can be trusted to satisfy recognition goals. As was mentioned earlier, in the tree recognition task SLS was given the goal of learning to find the image position of a tree to within an accuracy of three pixels. In twenty-one separate trials, SLS learned strategies for generating parabolic tree hypotheses, using a minimum distance classifier to verify goal-level hypotheses.

In each trial, SLS was trained on twenty images and tested on a single image. In general, strategies learned by SLS generated good hypotheses in eighteen of the twenty-one trials, for a success rate of 86%. The minimum distance classifier selected a correct hypothesis from this pool of goal-level hypotheses in seventeen of the eighteen cases for which correct hypotheses were generated. Overall, therefore, the system was able to satisfy the recognition goal by generating and verifying a correct goal-level hypothesis in seventeen of the twenty-one trials, or 81% of the time.

Table 4.2.5 summarizes the system's performance. The first three rows record SLS's performance in generating goal-level hypotheses. For each trial, the top row records the error in the best hypothesis generated, the second row shows how many goal-level hypotheses were generated, and the third row records how many of those goal-level hypotheses were correct to within the accuracy threshold of three pixels. The last two rows include goal-level classification, and thus present the system as an end-user would see it. The fourth row shows the error in the hypothesis selected as the best hypothesis by the minimum distance classifier, while the fifth row shows the system's confidence in its result in terms of the normalized

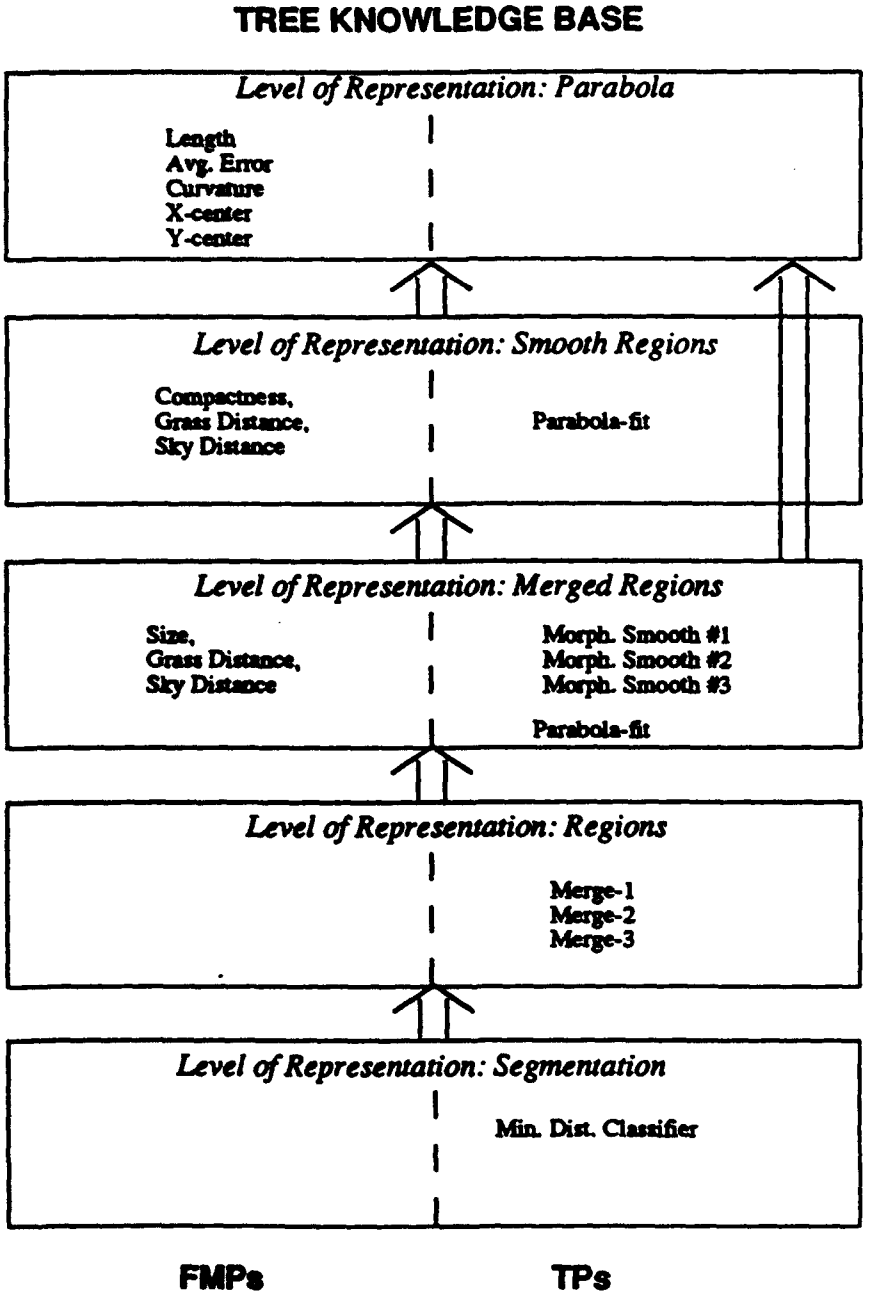

Figure 16: The tree recognition knowledge base. Feature measurement procedures (FMPs) are shown on the left hand side, while transformation procedures (TPs) are shown on the right. Every VP is shown at the level of representation of the hypotheses to which it can be applied.

euclidean distance in feature space between the best hypothesis and the learned prototype.

The third row of Table 4.2.5 emphasizes the extent to which SLS learns redundant strategies. The VPs in SLS's library – indeed, the algorithms in any computer vision toolkit – are prone to failure. To be robust, therefore, SLS must learn strategies that are redundant, so that if some VPs fail, others will still recognize the object. SLS minimizes redundancy in its strategies as much as possible while still successfully recognizing the object in every training image.

When learning to identify trees, SLS generally included all three parameterizations of the morphological smoothing TP, to ensure that at least one of them will produce a smooth region with a distinct peak at the center of the tree. In thirteen of the twenty-one trials, all three parameterizations produce quality regions, resulting in three correct parabola hypotheses. In four trials, however, one of the smoothing TPs failed to produce a region with a peak that the parabola fitting TP could identify, so that only two correct hypotheses were produced for those trials. Even more significantly, in trial eleven, two of the smoothing TPs failed, producing just a single correct hypothesis and demonstrating that including three smoothing TPs was necessary.

Trials fourteen, nineteen and twenty, on the other hand, show that SLS's strategies can fail, despite their redundancy. In these three trials, and only in these three trials, SLS learned strategies that included just two smoothing TPs. An *a posteriori* analysis shows that for three of the training images, only one of the smoothing TPs leads to a correct hypothesis. (For all other training images, at least two of the three versions of smoothing generate high-quality hypotheses.) As a result, on the trials in which one of these images was left out of the training set, the most efficient strategy for satisfying all of the examples included in the training set was to use only the other two versions of smoothing in the strategy. Of course, in a "leave one out" testing scheme, the image not included in the training set is the one used for testing, so the learned strategy failed on these three tests.

4.2.6 Efficiency Results

Table 4.2.5 addresses the robustness of the strategies learned by SLS, but not their efficiency. Table 2 shows SLS's expected run-time (in seconds, rounded to the nearest whole second) for the strategy learned in each trial as well as the actual run-time when the strategy was applied to a test image. On any given trial, the discrepancy between the predicted and actual run-times is quite large. On average, however, the predicted run-times are within one percent of the actual run-times. This reflects the average-case nature of expected costs. The actual cost of recognizing an object in an image depends critically on the contents of the image, but as long as the training images are indicative of the test domain the average cost of recognition can be estimated. Indeed, it is remarkable the predictions were accurate to within one percent. Another indication that the expected costs are accurate is that the expected cost exceeds the actual cost in eleven on twenty-one trials, or almost exactly fifty percent of the time.

Table 1: Results of twenty-one trials of learning to recognize the image position of a tree, with a minimum distance classifier for goal-level verification. The top row of the table shows the error in the image position (measured in pixels) of the best parabola hypothesis generated for each trial. The second row is the number of goal-level hypotheses produced by SLS's strategy, and the third row records how many of those hypotheses were within the accuracy threshold of the recognition goal (three pixels). The fourth row shows the positional error (again in pixels) of the hypothesis selected by the minimum distance classifier, while the bottom row shows the normalized euclidean distance in feature space between the best hypothesis and the object prototype learned by the minimum distance classifier. The averages of each row are shown in the last column.

Trial	Best Hyp.	Hyp. Count	Corr. Count	Sel. Hyp.	Distance
1	1.28	11	3	1.73	1.09
2	0.75	10	3	1.04	1.34
3	0.38	15	3	0.38	2.27
4	0.06	13	3	0.54	1.36
5	0.00	18	3	0.38	2.14
6	0.10	15	3	0.10	1.85
7	0.00	13	3	0.33	2.14
8	1.39	17	3	1.69	1.22
9	0.50	17	3	0.85	1.26
10	0.02	15	2	0.02	1.43
11	2.08	15	1	6.59	3.11
12	1.96	20	2	1.96	2.00
13	0.49	18	3	1.32	1.81
14	4.23	21	0	6.51	2.11
15	0.48	11	3	0.48	4.69
16	0.69	21	2	0.69	2.48
17	0.87	22	3	1.82	2.54
18	0.83	20	3	0.82	1.09
19	18.47	10	0	18.48	5.98
20	7.40	9	0	7.40	4.54
21	0.30	15	2	0.63	3.51
Avg.	2.01	15.5	2.29	2.56	2.38

Table 2: Timing results for the twenty-one tree recognition trials. The first row shows the expected cost (in seconds, rounded to the nearest whole second) of applying the strategy, as predicted by SLS. The second row shows the actual cost. Although the difference between expected and actual run-time for any given trial is quite high, the average expected run-time matched the average actual run-time to within one percent.

trial	1	2	3	4	5	6	7	8	9	10	11
Exp. Cost	459	463	463	440	450	463	460	452	451	458	459
Act. Cost	242	255	269	703	538	269	284	559	520	315	350
trial	12	13	14	15	16	17	18	19	20	21	Avg.
Exp. Cost	454	454	701	461	444	444	445	331	335	430	453
Act. Cost	436	420	824	301	627	626	649	211	567	626	457

4.3 Building Recognition from An Approximately Known Viewpoint

In the second demonstration, SLS learns to recognize the Marcus Engineering building, which is the red brick building immediately to the left of the tree in Figures 13 and 14. This time, however, the goal is to determine the three-dimensional location and orientation of the building relative to the camera, rather than the image position of its projection. By finding the three-dimensional pose of an object relative to the camera, recognition strategies can determine the position of a mobile vehicle from a single landmark, rather than having to triangulate among multiple landmarks, thus permitting landmark-based navigation in barren domains with few landmarks, or in environments that are only partially modeled.

4.3.1 Training Images

The strategies for recognizing Marcus are learned from the same set of training images that were described in Section 4.2.1, including the images in Figures 13 and 14. As discussed there, the images display four degrees of freedom, three that involve the position of the building and a fourth that determines it orientation. As before, SLS is tested by training it on twenty images and testing on a twenty-first, a process that is repeated twenty-one times until every image has been used as the test image.

The "ground truth" positions and orientations of the building were determined by manually matching image points to model points and applying Kumar and Hanson's algorithm [23] to determine the building's pose relative to the camera. The training signal is therefore composed of errorful pose estimates, rather than true positions. However, Kumar and Hanson's results suggest that, with correct correspondences, their algorithm produces pose estimates that are extremely accurate when compared to the relatively lax error thresholds in the recognition goal (see next section). The estimated poses can therefore reasonably be used as a training signal.

4.3.2 Recognition Goal

The recognition goal for this exercise is to find the pose of Marcus Engineering relative to the camera. Pose hypotheses are represented as rotation matrices with translation vectors, in the traditional

$$P' = RP + T$$

representation, where R and T are the rotation matrix and translation vector that transform a set of points P in the model's coordinate system into a set of points P' in the camera's coordinate system. Unfortunately, errors expressed in terms of R and T tend to be unintuitive, since if an object is rotated slightly about its center, this will be represented as a rotation about the focal point, counteracted by a large translation[9]. It is helpful, therefore, to express the error tolerances in a different representation.

Since the pose of the building has only four degrees of freedom, the tolerance thresholds in the recognition goal are expressed in terms of scale, image position, and object angle. These parameters reflect the fact that the pose of the building can be expressed as a vector from the focal point to any known point on the building, plus a horizontal rotation about the known point. (Remember that the building has no tilt or roll relative to the camera.) Errors in the positional vector are expressed as an error in length, measured as a percent of the true camera-to-object distance, and an error in position, measured as an angle (Since we are interested in the magnitude of the orientation error, not its direction, this can be written as a scalar.) Errors in the rotation of the object are also represented as an angle, this time about the vertical axis.

The error thresholds for this exercise are that the position of the building must be correct to within one degree of image angle and ten percent depth, and the orientation of the building must be correct to within five degrees. These thresholds require that the hypothesized pose be highly accurate with respect to the building's image position and reasonably accurate in the building's orientation, but only approximate in depth. Figure 17 shows an example of a pose that satisfies these criteria, in this case the building pose identified by SLS's strategy in the first of twenty-one trials (see Section 4.3.4).

4.3.3 The Knowledge Base

A knowledge base for three-dimensional recognition is considerably more complex than a two-dimensional recognition knowledge base. As before, the knowledge base includes visual procedures for generating and verifying region and region set hypotheses, although parabola hypotheses are no longer needed. In their place, the knowledge base uses image point and image line hypotheses, and sets thereof, which better represent the structure of a building.

The knowledge base includes many visual procedures for extracting and grouping two-dimensional representations such as points and lines. Lines can be extracted using the edge-linking algorithm of Boldt and Weiss [4], and regions can be extracted by the algorithm described in Beveridge, et. al. [3]. Regions that match an expected color and texture can be selected from a region segmentation by a multivariate decision tree, as described by Brodley

[9]The size of the counteracting translation is a function of both the extent of the rotation and the distance from the object center to the focal point.

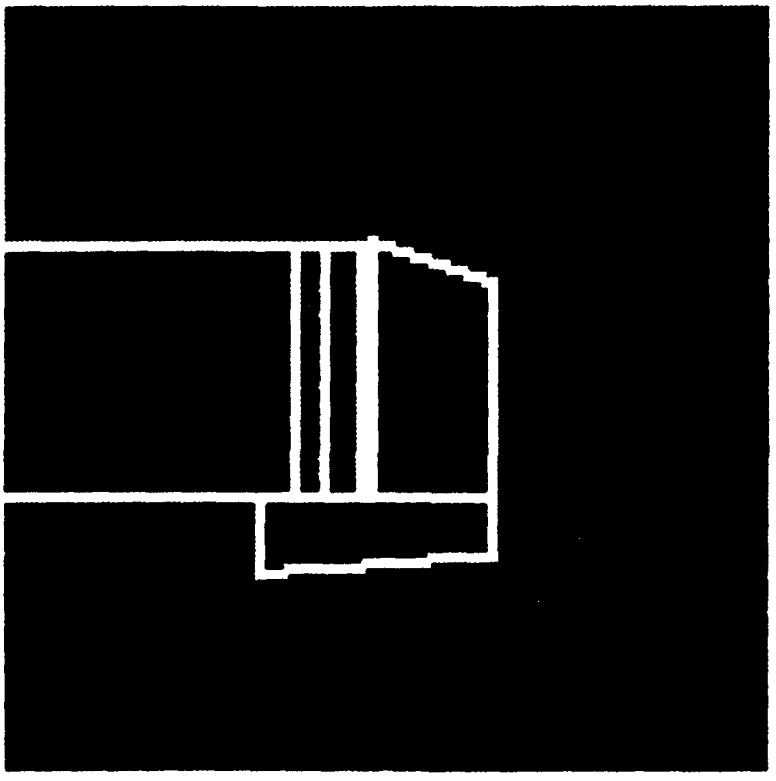

Figure 17: A correct pose from one trial of the Marcus recognition strategy. The pose shown here was generated and verified on the first of twenty-one trials, and is off by 1.8% in depth, 4.79 degrees in the the orientation of the building, and 0.16 degrees in the image location of the building.

and Utgoff [5]. (This algorithm is included twice in the knowledge base with two different parameterizations, one designed to select red brickface regions, the other highly textured window regions.) Nearby regions can be grouped by a region merging TP, while another TP groups lines that intersect a given region. Nearby lines that are parallel, collinear or orthogonal can be grouped according to the relations defined by Reynolds and Beveridge [28]. (All of the grouping VPs are implemented using the facilities of the ISR database system [6].) Image points are extracted either by finding trihedral junctions of lines, or by computing the convex hull of a pencil of lines.

In addition, new representations capable of supporting three-dimensional reasoning are introduced. Orientation hypotheses represent the orientation, but not location, of a plane in space, while planar surface hypotheses specify both the orientation and location of a plane. Most importantly, transformation hypotheses represent a coordinate transformation from one coordinate system to another, represented as a rotation matrix and a translation vector. Transformation hypotheses determine the pose of a modeled object by giving the transformation from the object model coordinate system to the camera coordinate system, and are the goal-level hypotheses in this demonstration.

Three dimensional hypotheses are generated and manipulated by geometric visual procedures. Collins and Weiss [11] provide an efficient TP for grouping line segments into *pencils*, which are sets of lines that meet at a common point of intersection. Vanishing point analysis [11] infers the orientations of planes in space by assuming that the image lines in a pencil are the projections of parallel lines in space. Another approach to inferring the orientation of an object in space is to find trihedral junctions of line segments first, and then use the perspective angle equations of Kanatani [22] to infer the orientations of the planes, assuming the lines form right angles, like the corner of a building.

The distance from an object to the camera can be estimated when the size of the object is known. In the case of Marcus Engineering, a wire-frame model of the object has been built from blueprints, as shown in Figure 18. Two parameterizations of the scaling TP are available in the knowledge base, one that estimates distance based on the apparent width of a window and the estimated angle of the building face, and a second that estimates distance from the height of the building using a direct inverse relationship of size to distance. (Note that since the images have zero tilt, the orientation of the building face is not needed to estimate distance from the building's height). Of course, since any two points on the object model can serve as compile-time parameters to a scaling TP, many other parameterizations of the scaling TP could be included in the knowledge base.

4.3.4 Reliability Results

Table 4.3.4 summarizes the results of twenty-one trials of learning to recognize the pose of Marcus Engineering from an approximately known viewpoint. The right side of the table shows the errors in the best goal-level hypothesis generated, even if this hypothesis was never verified, while the right side shows the errors in the goal-level hypothesis verified by the minimum distance classifier. The verified pose for trial number one, which is also the best pose generated for that trial, was shown earlier in Figure 17.

Pose errors in Table 4.3.4 are measured in terms of the length and orientation of a vector

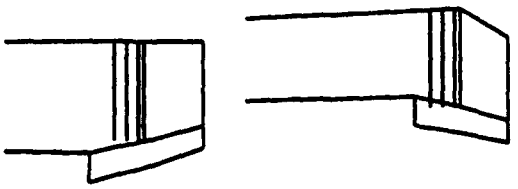

Figure 18: A wire-frame model of the Marcus Engineering Building, copied from its blueprints.

from the focal point to the corner of the building, and the rotation of the building. More precisely, the error in the position of the building is measured as 1) the error in the distance to the building, measured as a percentage of the true distance, and 2) the image position of the building, measured by the angle between the true vector from the focal point to the building corner and the estimated vector (labeled "Im Pos" in Table 4.3.4). The error in the building's orientation is measured as the angle in the horizontal plane between the estimated orientation of a building face and its true orientation (labeled "Rot." in Table 4.3.4).

The most striking feature of Table 4.3.4 is the result of trial sixteen. The strategy learned by SLS in trial sixteen did not generate a single goal-level hypothesis, either correct or incorrect, for the test image. An *a posteriori* analysis reveals that in twenty of the twenty-one images, the corner of the building is marked by a trihedral junction of image lines. In one image, however, noise eliminates one of the three lines. As a result, when the image without the trihedral junction is removed from the training set and used as the test image, SLS learns a strategy that relies entirely on finding trihedral junctions. The strategy does not succeed in finding any trihedral junctions in the test image, however, and therefore generates no goal-level hypotheses. Ironically, in the other twenty trials, the training sets include the case in which trihedral junctions fail, and therefore the other twenty strategies all include redundancy to account for the possibility of trihedral failure, and this redundancy is never needed for the test images to which they are applied.

Trial sixteen is the only case in which the strategy learned by SLS fails to generate a correct goal-level hypothesis, giving it a higher success rate at generating 3D building hypotheses (95%) than 2D tree hypotheses (86%). This improvement can be attributed to the geometric reasoning VPs, which in general are less *ad hoc* and more reliable than the region-based VPs used in the previous exercise. In trials fifteen and twenty, however, SLS verified the wrong hypothesis, giving it an over-all success rate of 86%.

4.3.5 Timing

Although the knowledge base for three-dimensional recognition is more complex than the two-dimensional knowledge base, and involves several more levels of representation, SLS is still able to predict the expected cost of its strategies accurately. Table 4.3.5 shows the expected cost for each strategy, as well as the actual cost when the strategy was applied to a

Table 3: Results of twenty-one trials of learning to recognize the pose of the Marcus Engineering Building. The left side of the table shows the errors in the best goal-level hypothesis generated for each trial, while the left side shows the errors in the hypothesis verified by the minimum distance classifier. Errors are specified in terms of the parameters discussed in Section 4.3.2, namely: 1) error in distance from the object to the camera, expressed as a percentage of the true distance from the object to the camera; 2) error in image position, measured in degrees; and 3) error in the building's orientation, measured in degrees.

	Best Generated Pose			Selected Pose		
Trial	Dist.	Rot.	Im Pos	Dist.	Rot.	Im Pos
1	1.81	4.79	0.16	1.81	4.79	0.16
2	1.34	0.82	0.05	1.34	0.82	0.05
3	1.18	1.33	0.11	2.74	1.33	0.09
4	0.69	1.40	0.13	1.97	1.40	0.13
5	2.64	2.33	0.10	2.64	2.33	0.10
6	0.73	6.95	0.15	0.73	6.95	0.15
7	0.27	1.16	0.05	6.58	1.16	0.07
8	2.33	0.07	0.08	2.33	0.07	0.08
9	1.01	3.58	0.20	1.69	3.58	0.20
10	1.39	1.65	0.04	2.07	1.65	0.05
11	0.50	3.27	0.08	2.37	3.27	0.25
12	2.24	4.48	0.25	2.24	4.48	0.25
13	2.07	1.54	0.04	2.07	1.54	0.04
14	0.36	1.63	0.09	0.36	1.63	0.09
15	2.13	2.58	0.21	11.23	2.58	0.21
16	–	–	–	–	–	–
17	4.44	6.21	0.13	4.44	6.21	0.13
18	1.07	1.75	0.09	1.77	1.76	0.16
19	0.41	3.83	0.07	1.07	3.83	0.07
20	0.92	2.50	0.03	14.64	2.50	0.03
21	1.18	4.95	0.13	2.26	4.95	0.13

Table 4: Timing results for the twenty-one Marcus Engineering trials. The first row shows the expected cost (in seconds, rounded to the nearest whole second) of applying the strategy, as predicted by SLS. The second row shows the actual cost.

Trial	1	2	3	4	5	6	7	8	9	10	11
Exp.	82.9	84.7	84.2	78.6	85.1	85.0	84.9	85.0	85.2	85.3	84.6
Act.	107.3	88.4	79.4	113.7	88.9	74.8	66.2	71.7	72.6	67.4	74.5
Trial	12	13	14	15	16	17	18	19	20	21	Avg.
Exp.	86.2	85.1	85.4	83.2	61.3	79.6	85.5	84.9	80.1	85.7	83.0
Act.	61.4	89.4	73.2	82.7	45.5	62.3	74.4	69.2	76.6	57.4	79.3

test image. While the variation in total time from one trial to the next is quite high, the average is once again close to the expected cost, with SLS overestimating the cost of its strategies by a mere 4.7 percent. Moreover, an *a-posteriori* analysis shows that SLS was highly accurate in estimating how often each visual procedure would be executed. SLS's overestimates were caused by VPs executing more quickly during testing than during exploration, due to variations in paging.

4.4 Recognizing Buildings from an Unknown Viewpoint

The final demonstration shows SLS learning strategies for recognizing a complex object from an unknown viewpoint. Despite its prevalence in the computer vision literature, viewpoint-invariant recognition is not a common visual task. Contextual knowledge about objects and viewers typically constrains the space of possible viewpoints, and even in this demonstration we will make a few contextual assumptions consistent with images taken from an autonomous vehicle, for example assuming that the camera is near the ground. Nonetheless, there are many situations where the relationship between the viewer and an object is unknown, and viewpoint-independent recognition strategies are needed. This demonstration was designed to show that SLS can learn strategies for this less common situation, too.

4.4.1 Training Images

In many respects, the training images for the final demonstration are similar to those of the earlier tests. The images are monocular, color images taken perpendicular to gravity (i.e., with no tilt or roll) from a few feet above the ground. But whereas the earlier images were taken from the same general area and pointing in the same direction, the new images were taken from random locations and orientations in a two hundred by three hundred foot quadrangle. The one thing all ten images have in common is that they all include part of the Lederle Graduate Research Center (LGRC), the L-shaped, multi-faceted building which houses the University of Massachusetts Department of Computer Science. Because of the differences in camera position and orientation from frame to frame, however, and because of the narrowness of the field of view, the images contain diverse and sometimes non-overlapping views of the building. Figures 19 and 20 show two of the ten images.

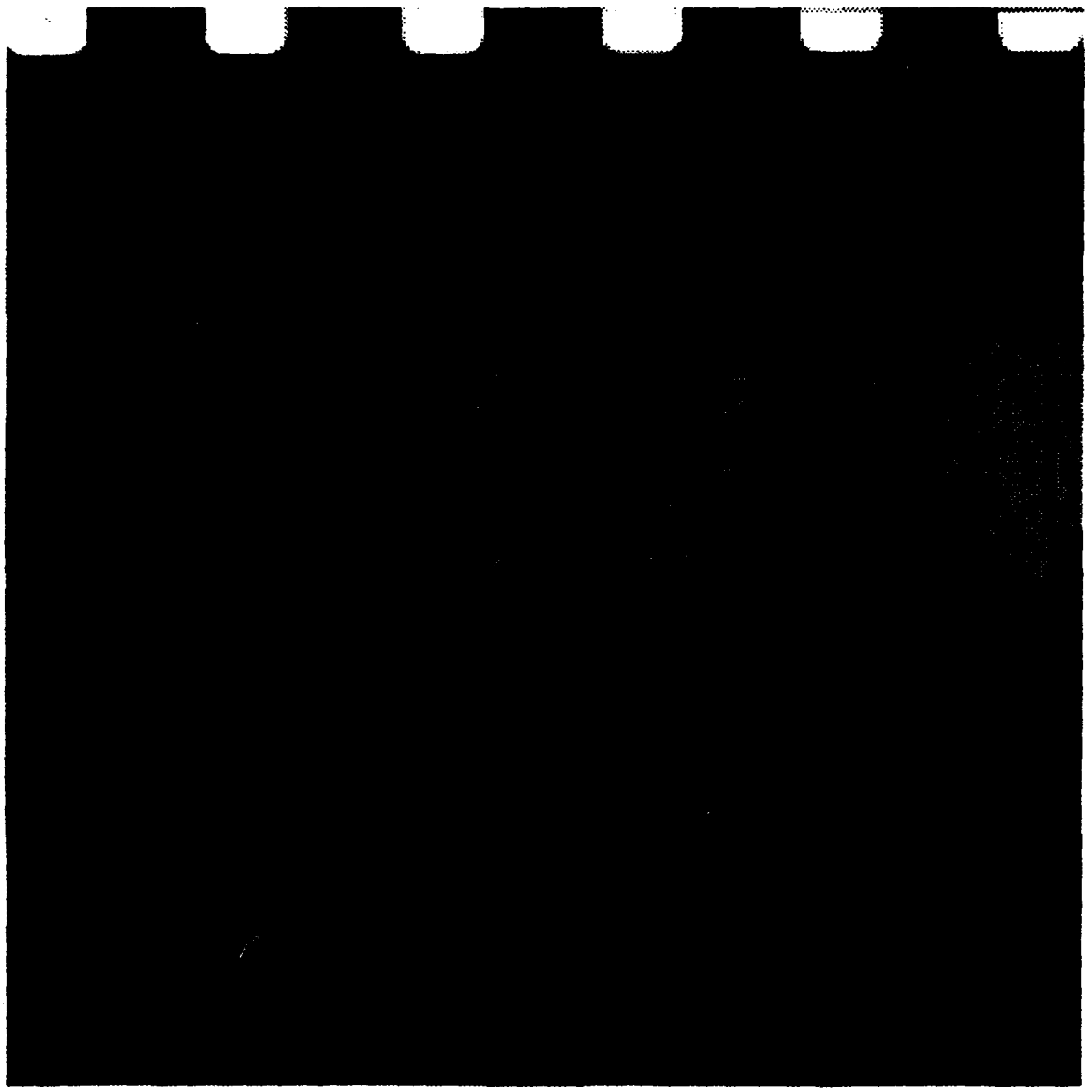

Figure 19: One of ten images of the Lederle Graduate Research Center (LGRC). The images were taken from random positions in the courtyard behind the building.

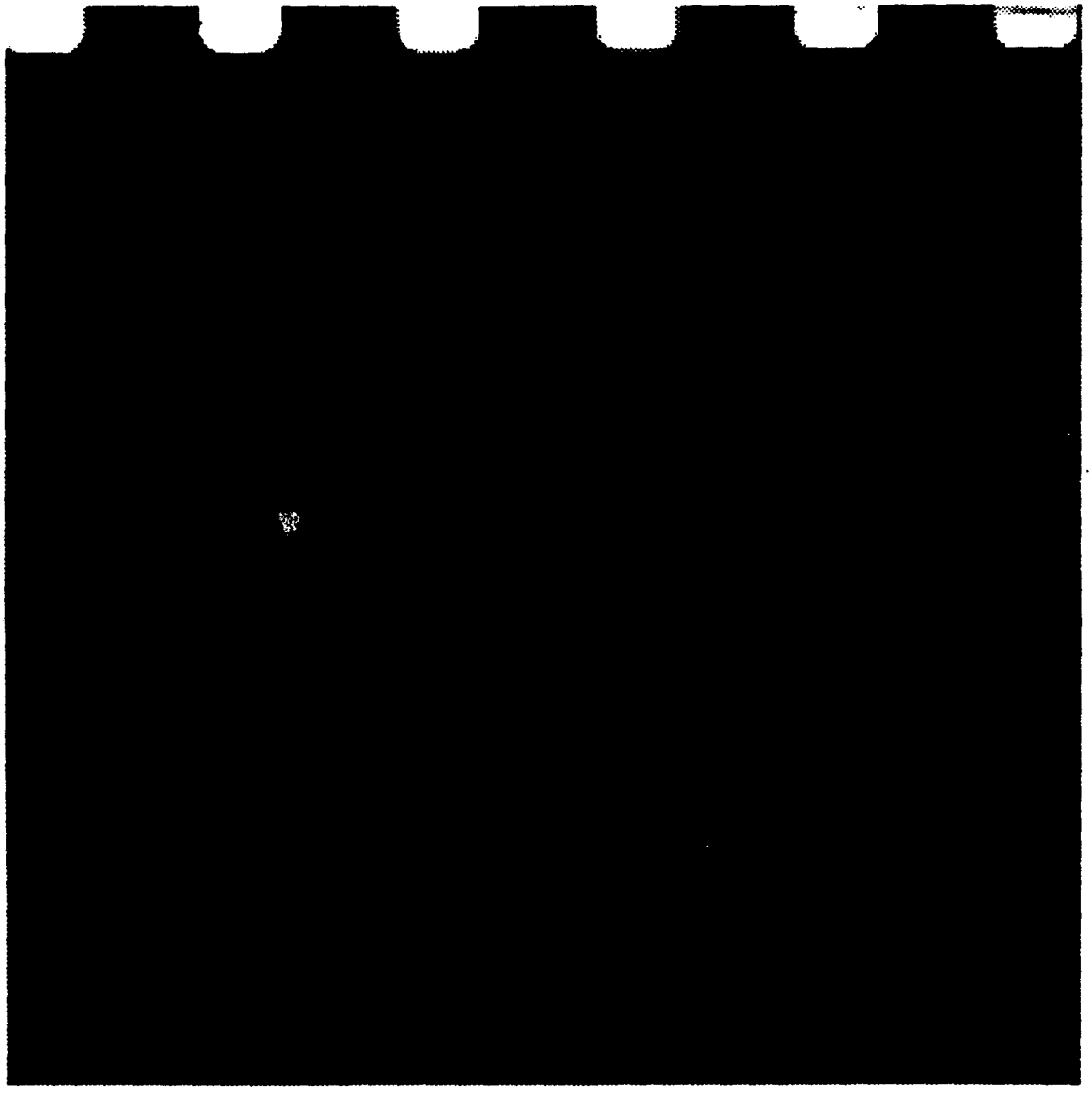

Figure 20: Another image of the Lederle Graduate Research Center (LGRC). Not only do the images of the LGRC view the building from different angles and at different scales, they also image different parts of building.

4.4.2 Recognition Goal

The recognition goal for this demonstration was to recover the position and orientation of the LGRC relative to the camera, plus or minus ten percent in scale, one degree in image position and ten degrees in pan angle. Ground truth positions for the LGRC were estimated by selecting correspondences between image points and points on a wire-frame model of the LGRC extracted from its blueprints. Kumar and Hanson's algorithm [23] was then applied to the hand-selected correspondences to establish estimates of the position and orientation of the LGRC in each image, estimates which served as a training signal. Figure 21 shows a correct pose hypothesis for the image shown in Figure 20, as generated in trial number nine.

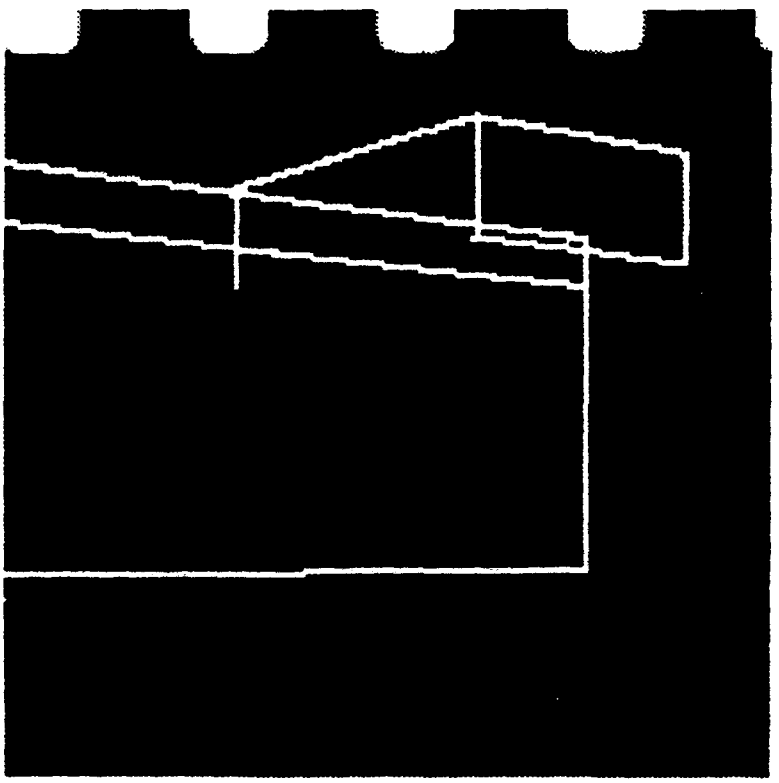

Figure 21: A correct pose from one trial of the LGRC recognition strategy. The pose shown here was generated and verified on the ninth trial, and is off by 5.93% in depth, 4.81 degrees in the the orientation of the building, and 0.11 degrees in the image location of the building.

4.4.3 The Knowledge Base

As one might expect, the knowledge base for recovering the pose of the LGRC is similar to the knowledge base for recovering the pose of Marcus Engineering. The LGRC knowledge base includes visual procedures for segmentation [3], region classification [5], line extraction [4], grouping lines into pencils [11], vanishing point analysis [11], perspective angle analysis [22], symbolic graph matching [29], and determining distance from scale.

Table 5: Results of ten trials of learning to recognize the pose of the Lederle Graduate Research Center (LGRC) from an unknown viewpoint. The right side of the table shows the errors in the best pose hypothesis generated, whether or not this hypothesis was verified by the minimum distance classifier. The left side of the table shows the errors in the verified hypothesis returned to the user by SLS. Errors are measured the same way as in Table 4.3.4 and discussed in Section 4.3.2.

	Best Generated Pose			Selected Pose		
Trial	Dist.	Pan	Im Pos	Dist.	Pan	Im Pos
1	3.06	2.12	0.98	53.01	2.12	27.31
2	32.02	8.17	17.37	32.02	8.17	17.37
3	–	–	–	–	–	–
4	2.52	4.08	0.12	5.32	4.08	0.12
5	–	–	–	–	–	–
6	1.04	5.83	0.17	1.04	5.83	0.17
7	–	–	–	–	–	–
8	8.22	8.72	4.42	12.96	8.72	2.93
9	5.93	4.81	0.11	5.93	4.81	0.11
10	3.24	6.93	0.15	3.24	6.93	0.15

One difference is that visual procedures for reasoning about shape are parameterized using the wire-frame model of LGRC rather than Marcus. The other major difference between the two knowledge bases is that the Marcus knowledge base was built for a single view. Consequently, whenever a significant point in the image is identified, it can be matched to the upper left corner of Marcus. In this demonstration, on the other hand, there is no single point on the building that is visible in all ten images. Instead, the knowledge base includes the coordinates of three model points, corresponding to the left corner of LGRC, the top of the stairwell and left corner of the housing for its air conditioning units, and tries to match one of them in order to fix the image position of the building.

4.4.4 Reliability

As before, SLS was tested with a "leave-one-out" methodology, this time with nine training images used on each of ten trials. Table 4.4.4 summarizes the results. As one would expect with only nine training samples for a complex task, the strategies learned by SLS prove less robust than in earlier demonstrations. In fact, it may be that robustness is inversely related to training set size, since halving the number of training images doubled the rate of failure.

Of course, the major impetus behind this demonstration was to show that SLS can learn to recognize objects from unknown, as well as known, viewpoints. The images in the test set not only view the LGRC from different angles, but some of them are of non-overlapping parts of the building. Nonetheless, SLS learns to recognize the LGRC partly by relying on VPs such as vanishing point analysis that are not view dependent, and partly by exploiting redundancy. In effect, SLS learns strategies that compensate for the multiple views by using one set of techniques to find the left-hand corner of the building, another set to find the stairwell, and yet a third set of techniques to find the right edge of the building. In fact,

Table 6: Timing results for the ten LGRC trials. The first row shows the expected cost (in seconds, rounded to the nearest whole second) of applying the strategy, as predicted by SLS. The second row shows the actual cost. The large variations in expected cost from one trial to the next are symptomatic of a learning algorithm that has not yet converged on a strategy after only nine training samples, and therefore produces very different strategies from one trial to the next.

Trial	1	2	3	4	5
Exp.	175.6	178.3	120.2	223.7	232.9
Act.	195.0	168.4	155.8	133.7	80.7

Trial	6	7	8	9	10	Avg.
Exp.	175.8	279.2	185.8	225.3	180.0	197.7
Act.	190.2	189.2	95.7	315.8	150.3	167.5

given that it has only a few examples of each view, it is remarkable that SLS did as well as it did, generating correct hypotheses in five of the ten trials (50%), and verifying a correct hypothesis in four trials (40%).

4.4.5 Timing

The lack of a sufficient training set impacts SLS's ability to predict the cost of its strategies as much as its limits its ability to produce robust strategies. Table 4.4.5 shows the expected and actual times for each trial, and unlike in previous trials the expected costs vary greatly from trial to trial. (In the earlier exercises, only the actual costs varied.) This is symptomatic of a learning strtaegy that has not yet converged on a consistent recognition strategy. Indeed, the expected cost of the learned strategies varies by almost a factor of two demonstrating that the strategy learned in one trial might be quite different from the strategy learned in another. It is not surprising, therefore, that the average actual cost is 15.3% below the average predicted cost across the ten trials.

4.5 Summary of Demonstrations

The demonstrations of two-dimensional and three-dimensional recognition from known and unknown viewpoints presented here are meant to convince the reader that SLS is more than a theoretical system, appropriate for recognizing abstract objects in synthetic images. SLS can learn practical recognition strategies for finding known objects in complex images. At the same time, these demonstrations are not intended as definitive experiments for testing the strengths and weaknesses of the system. SLS clearly needs to be tested on larger training sets, with more visual procedures, and under a wider variety of conditions. These tests demonstrate SLS in action, but do not probe its limits.

Unfortunately, we do not currently have the facilities for thoroughly testing SLS. In order to run exhaustive experiments, we need to collect large sets of images quickly and cheaply, and to generate training signals for those images. We also need the computational resources and disk space to be able to process and store hundreds of images.

4.6 Conclusion

Work on the Schema Learning System was initially motivated by the needs of its predecessor, the Schema System [12, 13]. The Schema System used appearance models and object relations to interpret complex natural scenes, and demonstrated how concurrent, special-purpose processes can cooperatively interpret images by exchanging tentative hypotheses. Unfortunately, it took so many hours of human labor to develop each schema that the Schema System was essentially limited to toy domains. As members of the Schema System research team, we were all too acutely aware of the cost of generating schemas and the limitations that implied.

Faced with a similar dilemma, researchers on the SPAM project [25, 18] and in Japan [24] sought to reduce the cost of knowledge base construction by building better tools and programming languages. In essence, they tried to finesse the knowledge acquisition problem by developing better software engineering tools to aid the human knowledge engineer.

We decided on a different approach. Rather than increase the speed with which knowledge engineers can craft a knowledge base, we decided to take the humans "out of the loop", by building systems that automatically learn to recognize objects. This approach depends on conceptualizing vision as a set of evolving skills rather than a single, fixed matching process. Each viewer, according to this paradigm, develops recognition strategies in response to its environment, optimizing and refining those visual skills that are used most often.

Of course, it would be arrogant, not to mention misleading, to imply that SLS as described here meets this goal. SLS is just a small step on which others, hopefully, will build. It would be nearly as arrogant to imply that SLS was, in any meaningful sense, "finished". This work argues for the importance of learning recognition strategies from libraries of visual procedures and presents one system for solving this problem, but there is still much work to be done on learning in vision before it becomes a reality. Happily, this line of research is being continued under a new contract from ARPA and Rome labs (see below).

5 Contributions (So Far)

This report presents a complete and implemented system for learning object recognition strategies. Unfortunately, the ideas underlying this system are sometimes obscured by the details of the data structures and algorithms themselves. One of SLS's main contributions is in its formulation of the problem of learning recognition strategies. Up until now, systems that combine learning and vision have either exploited a single learning technique, such as neural nets (e.g. ALVINN [26]), or else optimized strategies for exploiting a single vision technique, such as graph matching (e.g. Goad [16]). As a result, these systems have not taken advantage of the wealth of vision representations and algorithms developed over the last twenty years.

SLS, on the other hand, learns strategies for controlling libraries of visual routines and representations, integrating other researcher's results at the level of an automatic programming system. Whereas other systems try to replace twenty years of vision research, SLS exploits it.

Of course, SLS is not the first system to model vision in terms of visual procedures and hypotheses. The Schema System is just one of many earlier systems that applied blackboard technology to the task of computer vision. But these systems were *ad-hoc* and relied on humans to supply control heuristics, whereas SLS automatically learns strategies for integrating visual procedures into coherent strategies. Just as importantly, the principles by which SLS develops its strategies can be explicitly and scientifically analyzed, unlike the hand-built strategies of earlier systems.

SLS's second major contribution is in modeling vision as a sequence of alternating transformation and verification tasks. This idea is really just an application of the old generate-and-test paradigm to multiple levels of representation, but it can be viewed as an organizing principle for exploiting machine learning in computer vision. In terms of control, representational transformations are discrete steps that must be taken in sequence in order to match image data to abstract object models. Symbolic (often logic-based) machine learning techniques are well-suited to this task. Verification, on the other hand, is a filtering problem that can be solved by many methods, including neural networks, decision trees and traditional Bayesian classifiers. SLS was implemented with a DNF-based algorithm for selecting transformational procedures and univariate decision trees for verifying intermediate-level hypotheses, but the use of other symbolic inference and/or classification algorithms should be explored within the alternating transformation and verification model.

Finally, SLS works on real data, and as such is a proof of concept for learning recognition strategies. Clearly, a great deal more testing will be needed before it can be fielded as a practical system. Nonetheless, it shows that its algorithms and representations do work, at least on small sets of real data.

References

[1] Aloimonos, J. "Purposive and Qualitative Active Vision," *Proc. of the DARPA Image Understanding Workshop,* Pittsburgh, PA., Sept. 1990. pp. 816–828.

[2] Arbib, M.A. *The Metaphorical Brain: An Introduction to Cybernetics as Artificial Intelligence and Brain Theory.* New York: Wiley Interscience, 1972.

[3] Beveridge, J.R., Griffith, J., Kohler, R.R., Hanson, A.R. and Riseman, E.M. "Segmenting Images Using Localized Histograms and Region Merging," *International Journal of Computer Vision,* 2:311–347 (1989).

[4] Boldt, M., Weiss, R. and Riseman, E.M. "Token-Based Extraction of Straight Lines," *IEEE Trans. on Systems Man, and Cybernetics,* 19(6):1581–1594 (1989).

[5] Brodley, C.E. and Utgoff, P.E. "Multivariate Decision Trees," *Machine Learning,* to appear.

[6] Brolio, J., Draper, B.A., Beveridge, J.R. and Hanson, A.R. "The ISR: an intermediate-level database for computer vision", *Computer,* 22(12):22–30 (1989).

[7] Brooks, R.A. "A Layered Robust Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation,* 2(1) (March, 1986) pp. 14–25.

[8] Camps, O.I., Shapiro, L.G. and Haralick, R.M. *PREMIO: The Use of Prediction in a CAD-Model-Based Vision System,* Technical Report EE–ISL–89–01, Dept. of Electrical Engineering, Univ. of Washington, Seattle, WA., 1989.

[9] Chen, C-H, and Mulgaonkar, P.G. "Automatic Vision Programming," *CGVIP: Image Understanding,* 55(2):170–183 (1992).

[10] Cohen, P.R. and Feigenbaum, E.A. (eds). *The Handbook of Artificial Intelligence, Vol. 3.* Los Altos, CA.: William Kaufman, Inc., 1982.

[11] Collins, R.T. and Weiss, R.S. "Vanishing Point Calculation as a Statistical Inference on the Unit Sphere." *Proc. of the International Conference on Computer Vision,* Osaka, Japan, Dec., 1990, pp. 400–405.

[12] Draper, B.A., Collins, R.T., Brolio, J., Hanson, A.R. and Riseman, E.M. "Issues in the Development of a Blackboard-Based Schema System for Image Understanding," in [15] pp. 189–218.

[13] Draper, B.A., Collins, R.T., Brolio, J. Hanson, A.R., and Riseman, E.M. "The Schema System," *International Journal of Computer Vision,* 2:209–250 (1989).

[14] Draper, B.A. *Learning Object Recognition Strategies,* Ph.D. dissertation, University of Massachusetts, May 1993.

[15] Engelmore, R. and Morgan, T. (eds). *Blackboard Systems.*, London: Addison-Wesley, 1988.

[16] Goad, C. "Special Purpose Automatic Programming for 3D Model-Based Vision", in *Proc. of DARPA Image Understanding Workshop*, Arlington, VA., 1983. pp. 94–104.

[17] Hanson, A.R. and Riseman, E.M. "VISIONS: A Computer System for Interpreting Scenes," in *Computer Vision Systems,* Hanson and Riseman (eds.), N.Y.: Academic Press, 1978. pp. 303–333.

[18] Harvey, W.A., Diamond, M. and McKeown, D.M. "Tools for Acquiring Spatial and Functional Knowledge in Aerial Image Analysis", *Proc. of the DARPA Image Understanding Workshop,* San Diego, CA., Jan. 1992, pp. 857–873.

[19] Hillier, F.S. and Lieberman, G.J. *Introduction to Operations Research.* San Francisco: Holden-Day, Inc., 1980.

[20] Ikeuchi, K. "Generating an Interpretation Tree from a CAD Model for 3D-Object Recognition in Bin-Picking Tasks," *International Journal of Computer Vision* 1:145–165 (1987).

[21] Ikeuchi, K. and Hong, K.S. "Determining Linear Shape Change: Toward Automatic Generation of Object Recognition Programs," *CGVIP: Image Understanding*, 53(2):154–170 (1991).

[22] Kanatani, K. "Constraints on Length and Angle," *Computer Vision, Graphics, and Image Processing*, 41:28–42 (1988).

[23] Kumar, R. and Hanson, A.R. "Determination of Camera Position and Orientation," *Proc. of the DARPA Image Understanding Workshop*, Palo Alto, CA., May 1989, pp. 870–881.

[24] Matsuyama, T. "Expert Systems for Image Processing: Knowledge-Based Composition of Image Analysis Processes" *Computer Vision, Graphics, and Image Processing*, 48:22–49 (1989).

[25] McKeown, D.M. Jr., Harvey, W.A., and Wixson, L.E. "Automating Knowledge Acquisition for Aerial Image Interpretation" *Computer Vision, Graphics, and Image Processing*, 46:37–81 (1989).

[26] Pomerleau, D.A., Gowdy, J. and Thorpe, C.E. "Combining Artificial Neural Networks and Symbolic Processing for Autonomous Robot Guidance," *Proc. of the DARPA Image Understanding Workshop*, San Diego, CA, Jan. 1992. pp. 961–967.

[27] Quinlan, J.R. "Induction of Decision Trees", *Machine Learning*, 1:81–106 (1986).

[28] Reynolds, G. and Beveridge, J.R. "Searching for Geometric Structure in Images of Natural Scenes," *Proc. of the DARPA Image Understanding Workshop*, Los Angeles, CA., Feb. 1987. pp. 257–271.

[29] Ullman, J.R. "An Algorithm for Subgraph Isomorphism," *Journal of the ACM*, 23(1):31–42 (1976).

[30] Ullman, S. "Visual Routines," *Cognition*, 18:97-106 (1984).